TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie

Studijní obor: 1802R007 – Informační technologie

Programování pomocí grafických symbolů

Programming using graphical symbols

Bakalářská práce

Autor: **Tomáš Horák**
Vedoucí práce: Ing. Tomáš Martinec, Ph.D.
Konzultant: Ing. Martin Vlasák

V Liberci 17. 5. 2012

Tady bude zadání

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu bakalářské práce Ing. Tomášovi Martincovi, Ph.D. za čas, který mi věnoval a odborné rady, které mi velice pomohly při vypracovávání této práce. Dále pak mé rodině, za vytvořené zázemí a podporu při studiu.

Abstrakt

Cílem práce je popsat dostupné jazyky, které umožňují tvorbu algoritmů pomocí grafických symbolů, a prostředí, v nichž jsou tyto jazyky použity jako prostředek pro programování. Dále bude v práci navrhnut vlastní vizuální jazyk a bude vytvořeno prostředí, které pomocí tohoto jazyka bude umožňovat tvorbu jednoduchých programů a generování zdrojového kódu v některém klasickém programovacím jazyce.

Klíčová slova

grafický symbol, programování, vizuální programovací jazyk, vývojové prostředí, vývojový diagram

Abstract

The goal of this work is to describe available languages, which allow the creation of algorithms using graphical symbols and the environments in which these languages are used as a means for programming. Also there will be designed own visual language and programming environment that will allow the creation of simple programs and the generation of source code in any conventional programming language by using designed visual language.

Keywords

graphical symbol, programming, visual programming language, development environment, flowchart

Obsah

Seznam obrázků

Seznam zkratek

FBD – function block diagram

LAD – ladder diagram

PLC – programmable logic controller

SDCC – small device C compiler

UML – unified modeling language

1 Úvod

Tvorbu programů má většina lidí spojenou s psaním dlouhých, pro laika nesrozumitelných kódů v nejrůznějších programovacích jazycích. Méně lidí ale ví, že existují i další způsoby programování. Jedním z nich jsou vizuální jazyky, využívající k zobrazení algoritmů nejrůznějších symbolů. Snad každý programátor se setkal ve svých začátcích s jedním typem vizuálního jazyka – vývojovými diagramy. Vývojové diagramy se používají pro výuku algoritmizace na středních, někdy i na základních školách. Mnoho žáků k nim získá odpor, protože se povětšinou kreslí na papír a jakákoliv úprava už nakresleného diagramu vede ke škrtání, gumování nebo překreslování, a to komfort při práci jednoznačně snižuje.

V teoretické části práce budou rozebrány příklady vizuálních jazyků, používaných pro výuku, hry, ale i v průmyslu a budou uvedena vývojová prostředí, v nichž se tyto jazyky používají pro tvorbu programů.

Prvním bodem praktické části bude sestavení vlastního vizuálního jazyka, který bude umožňovat tvorbu algoritmů, popřípadě programů a bude v maximální možné míře odpovídat některé známé normě.

Druhým bodem bude vytvoření vývojového prostředí, které umožní základní práci s dříve sestaveným systémem symbolů a ověří jeho použitelnost.

2 Existující jazyky

2.1 Diagramy funkčních bloků

Diagram funkčních bloků (dále jen FBD) se používá hlavně pro programování programovatelných automatů (dále jen PLC), popřípadě pro modelování kombinačních obvodů. Tvorbu FBD ošetřuje mezinárodní norma IEC 61131-3.

FBD se skládá ze vstupních pinů diagramu, které jsou většinou vlevo, vedle jsou konkrétní funkční bloky a nakonec výstupní piny. Jednotlivé funkční bloky, jakožto i celý diagram, jsou orientovány zleva doprava. Použité vstupy jsou s funkčními bloky spojeny spojnicemi, které mohou být opatřeny šipkou. Šipka je povinná, pokud je orientace spojnice jiná než zleva doprava. Každý blok má vlevo vstupy, kam se mohou připojit spojnice ze vstupů diagramu, nebo výstupů jiných bloků a vpravo výstup(y). Každý vstup nebo výstup může být negovaný, což je znázorněné kolečkem.

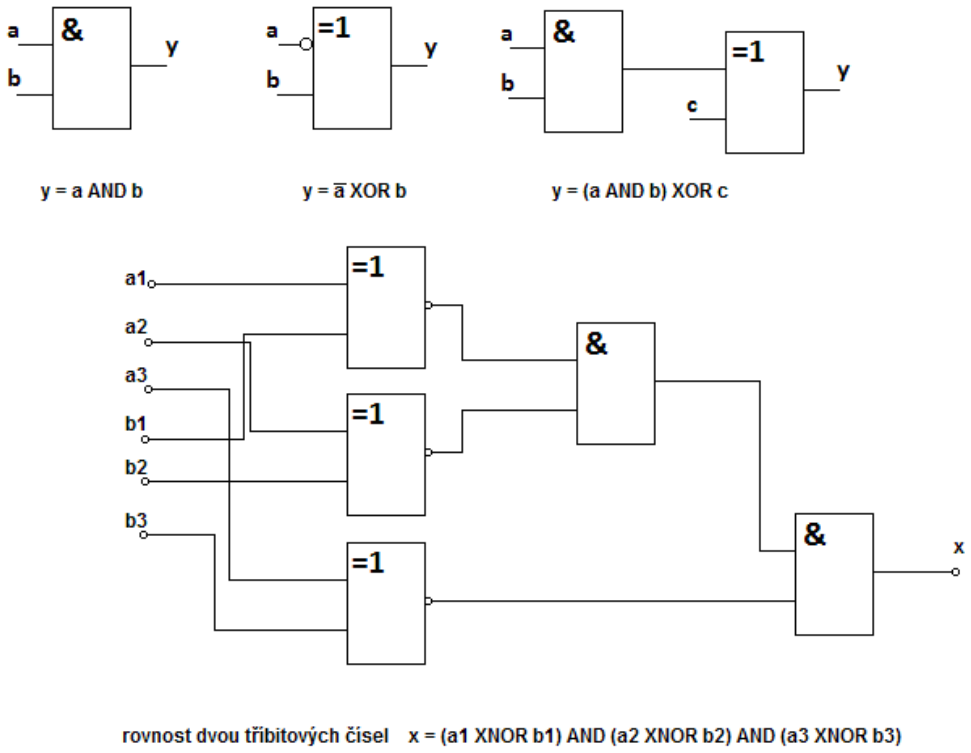
Norma rozlišuje dva druhy funkčních bloků, standardní, které bývají implementovány v každém vývojovém prostředí a speciální, které určí výrobce podle možností svého PLC. [1]

Standardní:

- AND
- NAND
- OR
- NOR
- XOR
- XNOR
- NOT

Speciální:

- zpožďovače
- čítače
- A/D převodníky
- atp.



Obrázek 1: Příklady funkčních bloků a jejich zapojení

2.2 Stavové diagramy

Stavovým diagramem se popisuje chování systému, který má konečný počet stavů. Tímto systémem může být například stavový automat. Stavový diagram je vyjádřen konečným počtem stavů, přechody mezi nimi, vstupy a výstupy. Algoritmus je stavovým diagramem vyjádřen jako posloupnost stavů.

Stav znázorňuje čas, ve kterém se nemění vstupní proměnné a automat čeká, nebo vykonává určitou akci. Automat v něm zůstává až do doby, než se změní vstupní proměnná. Stav může být znázorněn obdélníkem nebo kruhem s číslem či popiskem.

Přechod je spojnice mezi dvěma stavy, po které se automat pohybuje při změně vstupních proměnných, popřípadě po dokončení vykonávané akce. Směr pohybu je znázorněn šipkou. Volitelně může být součástí přechodu popisek

Návrh stavových diagramů je ošetřen podle specifikací jazyka UML.

Mealyho automat

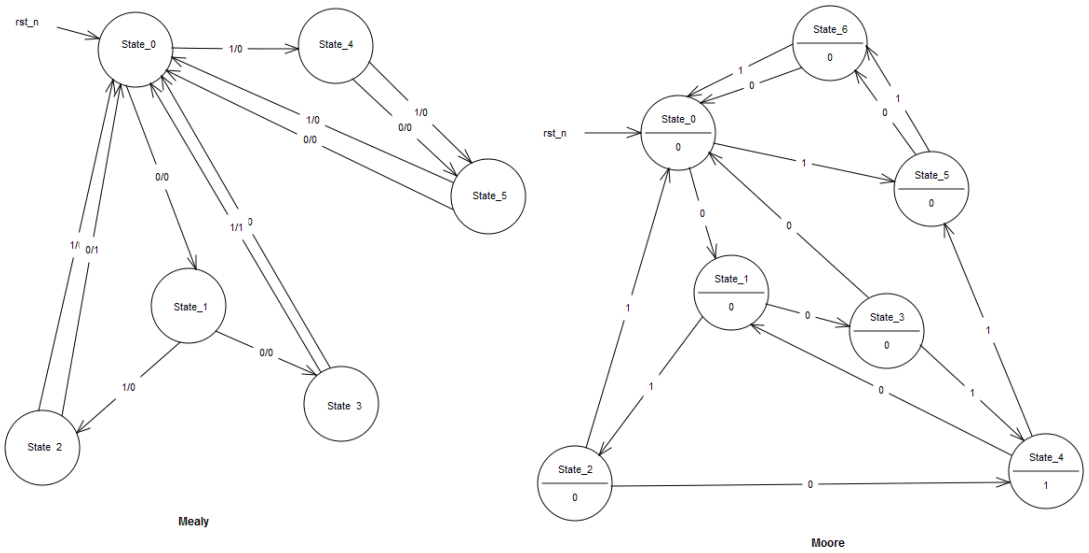
Mealyho automat je automat s konečným počtem stavů a výstupem, který je generován na základě stavu, ve kterém se nacházíme a vstupu. Výstup je tedy generován při přechodu mezi stavy. Každému automatu typu Mealy odpovídá obdobný automat typu Moore a lze je mezi sebou převádět.

Mooreův automat

Mooreův automat je také automat s konečným počtem stavů a výstupem. Na rozdíl od Mealyho automatu ale negeneruje výstupní hodnotu při přechodu, ale až v následujícím stavu. Výstup Mooreova automatu je tedy zpožděn a nezáleží na vstupní hodnotě, ale pouze na stavu, ve kterém se nacházíme.

Příklad

Automat rozeznávajícího trojice bitů ve tvaru 010 a 001.



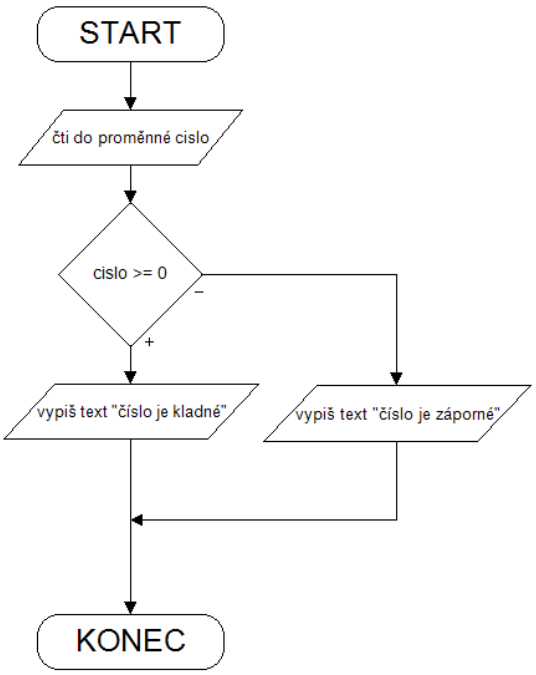
Obrázek 2: Příklad automatů Mealy a Moore

2.3 Vývojové diagramy

Vývojové diagramy se používají pro znázornění problémů zpracování informací a jejich řešení. Každá firma si může navrhnout svůj systém pravidel zápisu vývojového diagramu, ale obecné principy, jako je například tvar symbolů pro konkrétní typ operace, by měly v zájmu čitelnosti zůstat stejné. Diagramy se skládají ze symbolů, jež mají stanovený význam, dále pak z textu uvnitř symbolů, který vysvětluje jejich funkci a ze spojnic. Vývojový diagram je většinou orientován odshora dolů.

Univerzálnost vývojových diagramů spočívá v jejich použitelnosti na různých úrovních podrobnosti. To znamená, že celkový systém může být znázorněn diagramem ukazujícím jeho hlavní části a každá z částí může být zobrazena dalším diagramem s větší podrobností. Základní principy a tvary symbolů vývojových diagramů ošetřuje norma ČSN ISO 5807. [2] Tato norma specifikuje použití vývojových diagramů v těchto případech:

- Vývojové diagramy toku dat
- Vývojové diagramy programu
- Vývojové diagramy systému
- Síťové diagramy programu
- Diagramy zdrojů systému



Obrázek 3: Jednoduchý vývojový diagram

2.4 Scratch

Scratch je programovací jazyk vyvinutý v Massachusetts Institute of Technology primárně určený pro děti. Jeho cílem bylo umožnit dětem, ale i lidem, kteří nemají žádné zkušenosti s programováním, vyzkoušet si nejen tvorbu základních algoritmů, ale i prakticky použít vytvořený kód k ovládání různých grafických prvků nebo zvuků, podle možností použitého prostředí. Další vlastností, která dělá programování přitažlivější pro děti, je styl tvoření algoritmu. Symboly jsou vytvářeny přetahováním *drag and drop* a pouze přímé hodnoty jsou zadávány z klávesnice.

Původní program Scratch je šířen jako open source projekt, a proto vzniklo velké množství odvozených programů, které si vzaly Scratch jako základ. Konkrétní systém symbolů není ošetřen žádnou normou, takže se může lišit program od programu.

Algoritmus vytvořený v jazyce Scratch se nejvíce podobá kostkám stavebnice, které do sebe zapadají podle možností návaznosti funkcí. Jednotlivé druhy příkazů jsou od sebe odlišeny i barevně, což přidává výsledku na čitelnosti, ale i na pestrosti a přitažlivosti.

Obrázek 4: Počítání mocniny v jazyce Scratch

2.5 Další jazyky

Kromě zde uvedených jazyků samozřejmě existuje i mnoho dalších, jak už standardizovaných, tak i vyvinutých specifickou firmou pouze pro své specifické prostředí. Příkladem může být jazyk Ladder diagram (dále jen LAD), který je úzce spojený s jazykem FBD v prostředí Siemens LOGO!Soft. Algoritmus je vyjádřen

podobně jako elektrický obvod pomocí dvou vodičů, mezi nimiž jsou kontakty, cívky a speciální bloky propojené spojnicemi. Proměnné jsou vyjádřeny kontakty sepnuto = *TRUE,* rozepnuto = *FALSE*. Výstupy jsou vyjádřeny cívkami. Speciální bloky jsou vyjádřeny podobně jako v FBD obdélníkem. LAD je také ošetřen normou IEC 61131-3.

Dalším zajímavým jazykem je Microsoft Visual Programming Language. Tento jazyk byl vyvinut firmou Microsoft především pro ovládání nejrůznějších robotů, ale jdou s ním tvořit i jednoduché aplikace na platformu PC. Algoritmus je složen z bloků, které mají tvar obdélníků a jsou odlišeny barevně dle funkcí. Dostupné jsou základní bloky, jako vytváření proměnných, výpočty, podmínky, dále je k dispozici velké množství specializovaných bloků z různých knihoven a blok, který sám o sobě může obsahovat vnořený algoritmus, čímž se hlavní algoritmus graficky zjednoduší. Každý blok má vstupy a výstupy a jednotlivé bloky jsou spojeny spojnicemi.

3 Existující prostředí

3.1 LOGO!Soft

Prostředí LOGO!Soft je určeno k programování PLC LOGO! od firmy Siemens. Přístroje LOGO! najdou své uplatnění v průmyslu pro spínání ventilačních systémů, čerpadel, kompresorů, ale i jako spínače osvětlení, rolet nebo topení. Modul má 8 základních a 26 speciálních funkcí, výhodou je dokumentace i samotné prostředí v češtině.

- Výrobce: Siemens
- Internetová adresa: www1.siemens.cz/ad/current/index.php?ctxnh=3dc1f5a3fc
- Licence: Komerční, Demoverze zdarma
- Velikost instalačního balíku: 68MB (verze 6.0)
- Dostupné jazyky: FBD, LAD
- Výstup: simulovatelný model, kompilovaný program pro PLC

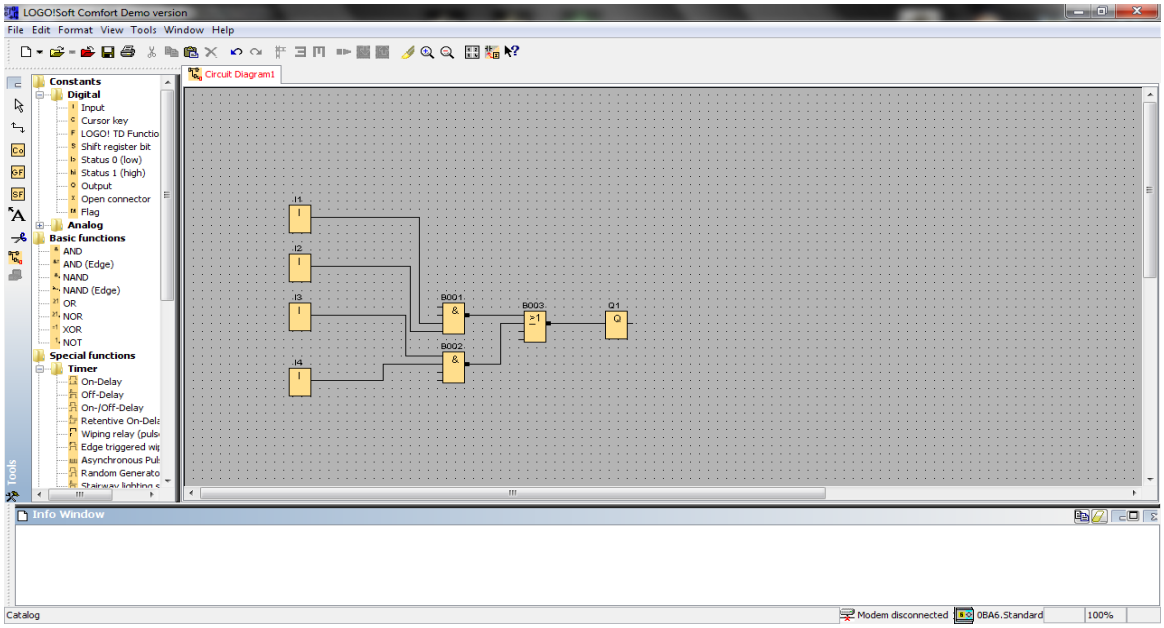
Prostředí LOGO!Soft je na první pohled velice přehledné a jednoduché. Vlevo je stromová struktura funkčních bloků obsahující:

- konstanty
- bloky pro prácí s digitálním vstupem a výstupem
- bloky pro prácí s analogovým vstupem a výstupem
- základní funkce – AND, OR, NOT, XOR, NAND,NOR
- speciální funkce
- časovače
- čítače
- bloky s analogovým vstupem
- ostatní

Pod stromovou strukturou pro výběr bloků se nachází okno s informačními hlášeními a celému programu dominuje pracovní plocha, na které se vytváří obvod.

Vlastní programování je uživatelsky přívětivé. Bloky i spoje se přichytávají k mřížce buď ručním klikáním na jednotlivé vstupy či výstupy bloků, nebo dvojklikem na vybraný vstup nebo výstup a následným vybráním bloku, který chceme spojit. Spoj

počítač vykreslí za nás. Nevýhodou je, že základní bloky (AND,OR…) mají pevně stanovený počet vstupů na 4.



Obrázek 5: Ukázka zapojení funkčních bloků v prostředí LOGO!Soft

3.2 Qfsm

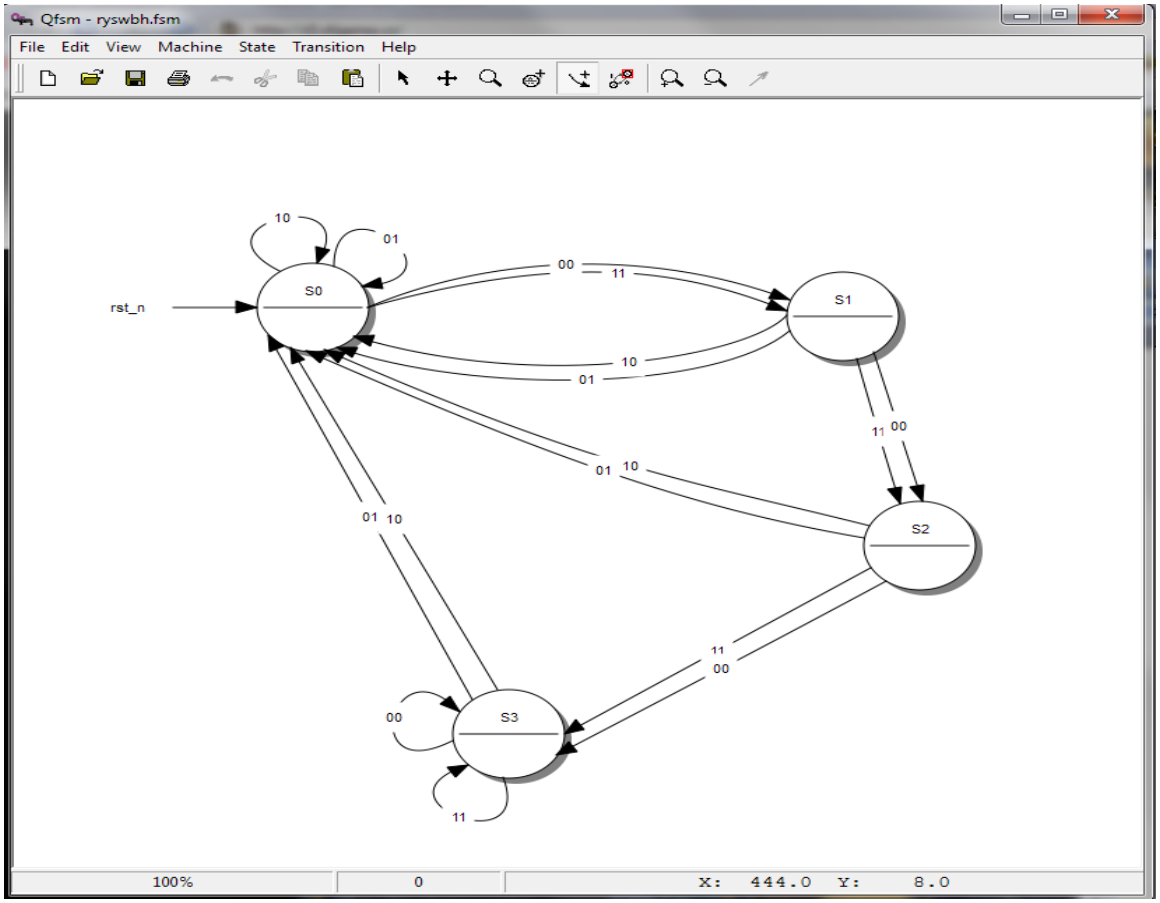
Qfsm je prostředí umožňující tvorbu stavových automatů typu Moore i Mealy.

- Autoři: Eugene Pivnev, Erwin Naroska, Nick Shaforostoff, Camille Decock
- Internetová adresa: http://qfsm.sourceforge.net/
- Licence: Freeware (Open Source)
- Velikost instalačního balíku: 6MB (verze 0.52)
- Dostupné jazyky: Stavové diagramy
- Výstup: Simulovatelný model

Prostředí je velice jednoduché a intuitivní, obsahuje pouze pracovní plochu a lištu s nástroji. Při vytváření nového projektu si zvolíme počet vstupů a výstupů (0 výstupů u automatu typu Moore, protože ty se určují až ve stavech), dále si můžeme vybrat z několika druhů šipek pro přechody, vybrat si font a napsat popisek programu. Samotné prostředí obsahuje jen ty nejnutnější ovládací prvky – práci s projektem, posunutí na pracovní ploše, lupu, přidání stavu a přechodu a start simulace.

Přidávání stavů je jednoduché, zvolíme si jméno, binární kód, u Moorova automatu výstup, (poloměr kružnice, šířku čáry a barvu) a popisek. Přidání přechodů

probíhá obdobně, volíme typ podmínky (binární, ASCII, volný text), vstupní hodnotu, (výstup u Mealyho automatu) a popisek. Simulace probíhá velice přehledně a je jednoduché otestovat správnost návrhu.



Obrázek 6: Ukázka stavového automatu v prostředí Qfsm

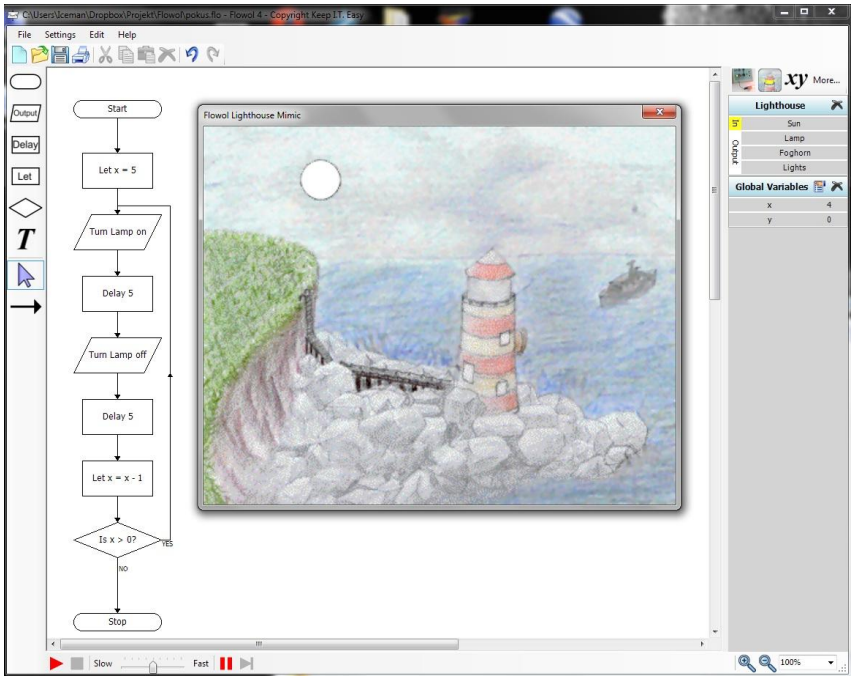
3.3 Flowol

Flowol je program zaměřený na výuku algoritmů spíše pro menší děti. Můžeme pomocí něho ovládat buď připojené mikropočítače a různé periferie, nebo pěkně zpracované obrázky prostředí zvané „*Mimic*“. V každém *Mimicu* je několik prvků, které lze algoritmem měnit. Na ukázce dole je to světlo majáku, zvuk majáku, slunce a mlha.

- Výrobce: Keep I.T. Easy
- Internetová adresa: www.flowol.com
- Licence: Komerční, zaměřeno na školy; testovací, časově omezená verze zdarma
- Velikost instalačního balíku: 37MB (verze 4.07)
- Dostupné jazyky: Vývojové diagramy
- Výstup: Model ovládající *Mimic* nebo připojený hardware

Prostředí je jednoduché, po spuštění programu si vybereme buď periferii, nebo *Mimic* s kterým chceme pracovat. Vlevo je výběr prvků digramu, který v prostředí tvoříme. Prvků není mnoho a jsou to klasické ovály pro začátek/konec programu a podprogram, výstup tj. nastavování prvků *Mimicu*, zpoždění, práce s proměnnými, podmínka, popisek a spojnice. Vpravo se nachází seznam ovladatelných prvků a seznam proměnných, uprostřed pracovní plocha. V samostatném okně se otevře grafika samotného *Mimicu*.

Na příkladu je diagram, který třikrát rozsvítí a zase zhasne světlo majáku se zpožděním 5s. Program bohužel neobsahuje symbol pro cyklus, ten se ale dá jednoduše nahradit pomocnou proměnnou, dekrementací a podmínkou na konci.



Obrázek 7: Ukázka vývojového diagramu v prostředí Flowol

3.4 Picaxe programming editor

Picaxe je mikroprocesor od stejnojmenného výrobce určený pro výuku, ovládání robotů, ale i velkých motorů, chladících zařízení apod. Mikroprocesor je dodávaný tak, aby se dalo nahrávání programu uskutečnit bez programátoru, pouze pomocí sériového kabelu a počítače s nainstalovaným vývojovým prostředím. [3]

- Výrobce: Revolution Education LTD
- Internetová adresa: www.picaxe.com
- Licence: Freeware

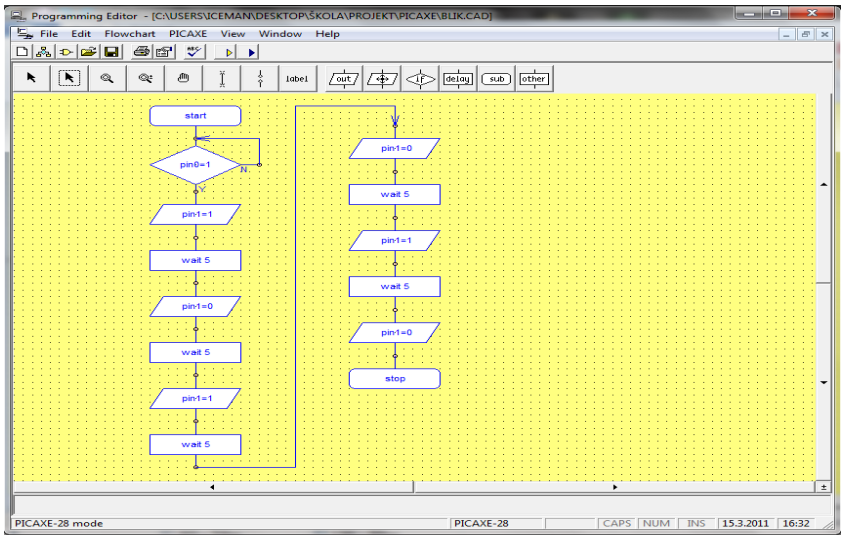
- Velikost instalačního balíku: 36MB (verze 5.5.0)
- Dostupné jazyky: BASIC, FBD, vývojové diagramy
- Výstup: Simulovatelný model, program pro nahrání do mikroprocesoru

Vývojové diagramy

Prázdná pracovní plocha pro vývojový diagram obsahuje pouze zakulacený obdélník jakožto blok *START* a mřížku k jednoduššímu uchycení jednotlivých bloků diagramu. Stavba vývojového diagramu se provádí jednoduchým vybráním žádaného bloku z nabídky a přetažením na pracovní plochu. K dispozici jsou:

- blok podmínky (kosočtverec) - buď pin mikroprocesoru nebo proměnná
- blok pohybu ve směrech, zastavení a nastavení rychlosti (kosodélník)
- blok výstupních funkcí (kosodélník)
- blok čekání (obdélník)
- blok pro podprogramy (zakulacený obdélník) - GOSUB, SUB, RETURN, STOP
- ostatní bloky (obdélník) - např. čtení do paměti, zápis do paměti, čtení adresy

Vybrané bloky na pracovní ploše se poté dají libovolně propojit pomocí tlačítka „*draw lines*“. Pokud je potřeba spoj rozvětvit, lze použít tlačítko „*add connection points*“. V praxi ale propojování ztěžuje fakt, že pokud jsou bloky moc blízko u sebe, spojnice se nekreslí správně. Dále prostředí obsahuje možnost přidání popisků, tlačítko pro pohyb v pracovní ploše, tlačítka pro přiblížení a oddálení a tlačítka výběru.



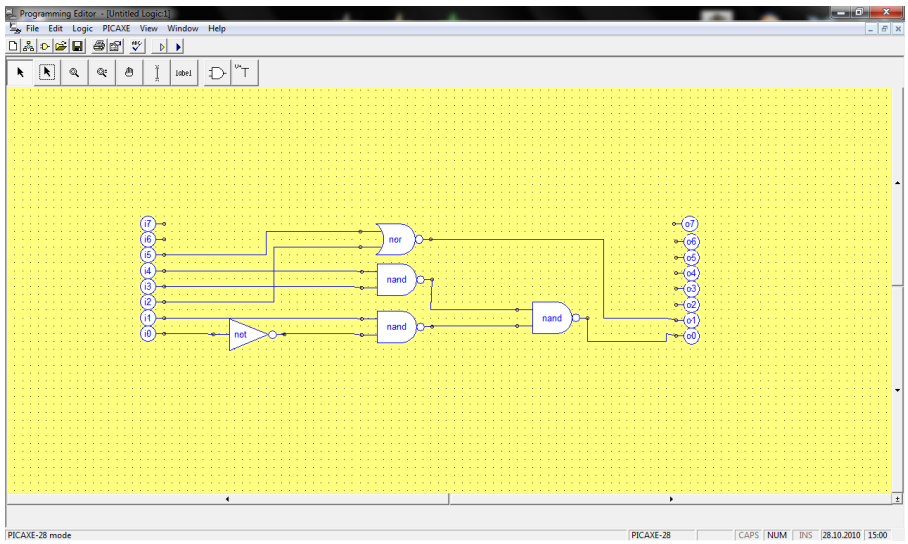
Obrázek 8: Ukázka vývojového diagramu v prostředí Picaxe

Na ukázkovém příkladu vidíme program, který pětkrát blikne diodou připojenou na *pinu1* po stisknutí tlačítka na *pinu0*. Program neobsahuje symbol pro cyklus, proto je i pouhé blikání se zpožděním docela dlouhá posloupnost symbolů. Samozřejmě je možné zjednodušení pomocí podmínky.

Logické bloky

Prázdná pracovní plocha obsahuje vlevo 8 pinů vstupu mikroprocesoru (i0 – i7) a vpravo 8 pinů výstupu (o0 – o7). Mezi ně se vkládají logické bloky. Na výběr jsou NOT, AND, NAND, OR, NOR, XOR, XNOR. Program používá americkou normu pro tvary symbolů. Dále lze na pracovní plochu vložit pouze napájecí napětí U+ a zem 0V. Jako u tvorby vývojových diagramů je možné vložit popisek. Nechybí ani tlačítko pro pohyb na pracovní ploše, tlačítka pro přiblížení, oddálení a výběr.

V praxi trpí programování pomocí logických bloků stejnými problémy jako programování pomocí vývojových diagramů. Další potíží je, že spoj nelze rozvětvit. To znamená, že na jeden výstup můžeme připojit pouze jedno hradlo. Spoje se občas nenakreslí rovně, ale mírně šikmo, takže pokud je výsledný logický obvod trochu složitější, vypadá nevzhledně.



Obrázek 9: Ukázka zapojení funkčních bloků v prostředí Picaxe

4 Zvolený systém symbolů

Cílem bylo zvolit takový systém symbolů, který by umožňoval tvorbu programů pro PC, ale také pro vývojovou desku Atmel AT89C51CC03 s procesorem 8051, a zároveň byl ošetřený nějakou jasně danou normou. Za těchto podmínek byly zvoleny vývojové diagramy podle normy ČSN ISO 5807 s určitými omezeními. Nespornou výhodou vývojových diagramů je, že většina programátorů se s jejich tvorbou setkala buď ve škole při výuce algoritmizace, kde jsou s oblibou používány jako grafické znázornění algoritmů nebo v nějakém konkrétním software, kde mohou být použity jako způsob programování. Další výhodou je jejich přehlednost a možnost použití na více úrovních abstrakce. To znamená, že můžeme celý systém popsat jedním jednoduchým diagramem, ale za každým jeho symbolem se může skrývat blok instrukcí, který může být podrobněji popsán dalším diagramem.

Vývojové diagramy budou použity pro generování kódu v jazyce C# pro programy na platformě PC a v jazyce C pro desku Atmel. Z toho vyplývá důležité omezení v jejich tvorbě. Posloupnost symbolů musí být taková, že každý symbol bude vyjadřovat pouze jednu instrukci a nebudou povoleny zpětné skoky, kdy by se program vracel do části, kde už jednou proběhl. Výjimkou jsou cykly, ale ani u symbolu cyklu není žádná spojnice, která by vedla tam, kde už program proběhl. Z toho vyplývá, že pro zachování jednoduchosti nebude dostupný cyklus *WHILE* s podmínkou na konci.

Norma ČSN ISO 5807 [2] uvádí 5 oblastí použití vývojových diagramů. Pro tuto práci bude důležitý Vývojový diagram programu, ten je definován jako diagram zobrazující posloupnost operací programu a skládá se z:

1) symbolů zpracování pro vlastní operace zpracování, včetně symbolů definujících stanovený tok, který má být dodržen při zachování logických podmínek;
2) spojnic indikujících tok řízení;
3) zvláštních symbolů pro usnadnění čtení a zápisu vývojového diagramu.

Dále uvádí velké množství symbolů. Pro účely této práce budou stačit jen tyto základní:

1) Mezní značka
2) Spojnice
3) Data
4) Zpracování
5) Příprava

6) Rozhodování

Každý symbol už svým tvarem identifikuje funkci, kterou představuje. Tato funkce je ještě více konkretizována krátkým textem uvnitř. Podle normy by měly mít všechny symboly v diagramu jednotnou velikost, pro lepší čitelnost toto není dodrženo a je přidána možnost velikost měnit.

U spojnic mezi symboly norma pouze uvádí možnost použít šipky pro zvýšení čitelnosti, což je dodrženo a každá spojnice má na konci šipku. Dále definuje připojování a také křížení spojnic, kdy nesmí docházet k žádné logické vazbě mezi spojnicemi, které se kříží. Připojování a křížení je rozlišeno šipkou v případě připojování, což je dodrženo.

4.1 Použité symboly

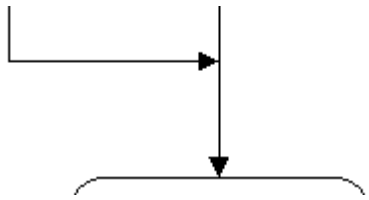
Mezní značka

- Tvar oválu
- Představuje začátek nebo konec programu, text uvnitř je „START“ nebo „KONEC“. Má pouze jeden vstup nebo výstup

Obrázek 10: Mezní značka

Spojnice

- Jednoduchá čára se šipkou na konci
- Představuje tok programu, šipka na konci směr. Může docházet ke spojování, ale ne k větvení.

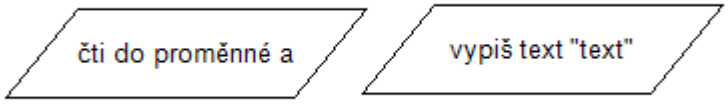


Obrázek 11: Spojnice

Data

- Tvar kosodélníku

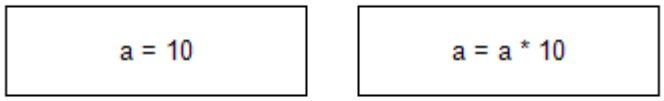
- Představuje vstup nebo výstup dat. Text uvnitř může být „čti do proměnné“ nebo „vypiš proměnnou“, „vypiš text“ a poté proměnná nebo u výpisu textová hodnota. Má jeden vstup a jeden výstup.



Obrázek 12: Data

Zpracování

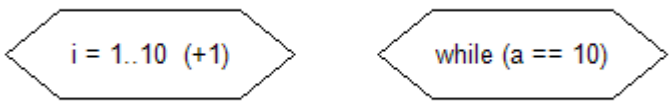
- Tvar obdélníku
- Představuje nějakou operaci s daty. Text uvnitř konkretizuje operaci. Dostupné jsou operace přiřazení, součet, rozdíl, součin, podíl, dělení modulo, inkrementace, dekrementace a generování náhodného čísla. Má jeden vstup a jeden výstup.



Obrázek 13: Zpracování

Příprava

- Tvar šestiúhelníku
- Dle normy tento symbol představuje přípravu, což může být například inicializace rutiny, úprava indexové proměnné nebo nastavení spínače. Pro zjednodušení bude použit jako identifikátor cyklu *FOR* nebo *WHILE*. Druhou odchylkou od normy bude zanedbání sekvenčního vstupu. Text uvnitř konkretizuje zvolený typ cyklu, v případě cyklu *FOR* určuje podmínku a práci s indexovou proměnnou a v případě cyklu *WHILE* pouze podmínku.

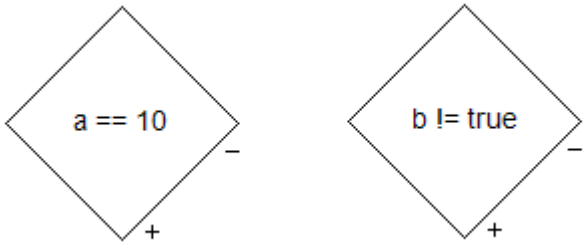


Obrázek 14: Příprava

Rozhodování

- Tvar kosočtverce
- Představuje podmíněné větvení programu. Text uvnitř vyjadřuje podmínku. Možnosti porovnání se liší podle typu porovnávané hodnoty, u čísel to jsou

rovnost, nerovnost, vetší, větší nebo rovno, menší a menší nebo rovno. V případě booleovských nebo řetězcových hodnot pouze rovnost a nerovnost. Symbol má jeden vstup a dva výstupy.



Obrázek 15: Rozhodování

4.2 Proměnné

Proměnné jsou důležitou součástí většiny programů, ale norma práci s nimi nijak nespecifikuje. V jazycích C# a C je nutné proměnnou nejdříve deklarovat na správný datový typ a bylo důležité rozhodnout, jestli pro tuto operaci použít nějaký symbol, nebo zvolit nějakou alternativu. Konečné rozhodnutí bylo takové, že proměnné se budou tvořit nezávisle na vývojovém diagramu a na začátku každého programu budou automaticky deklarovány a inicializovány na výchozí hodnoty.

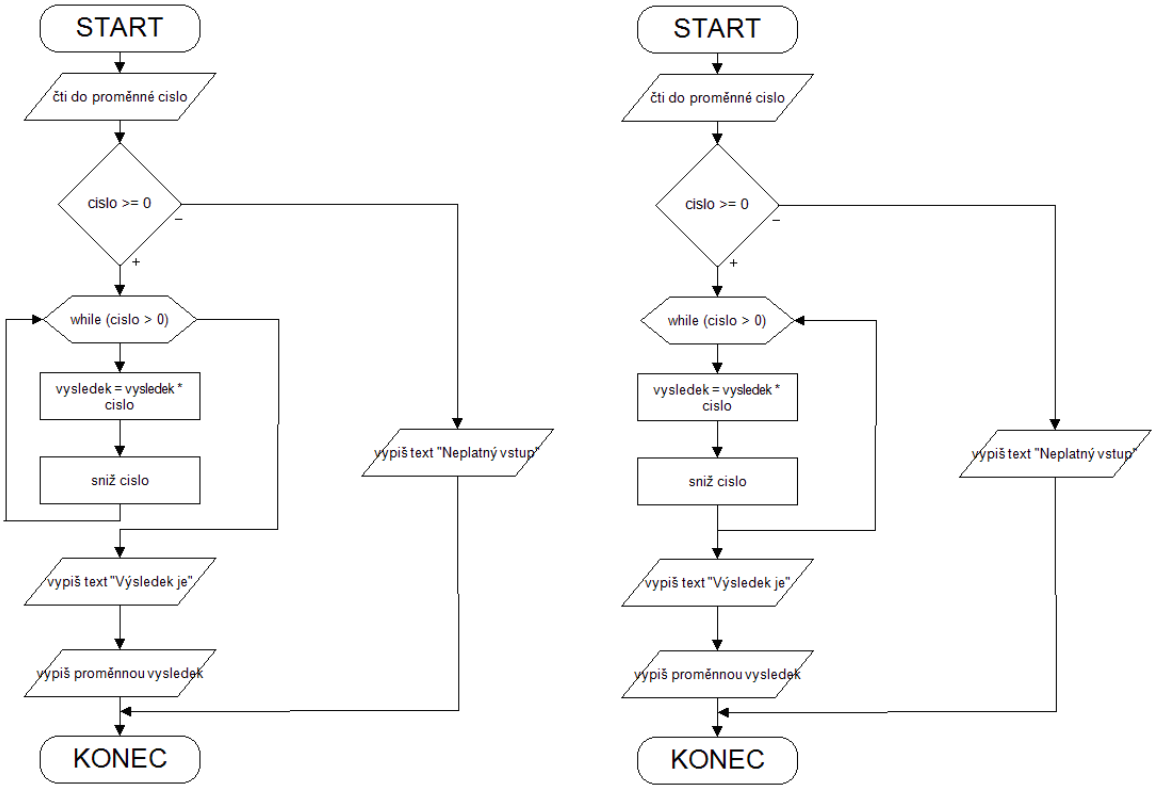
Další volbou byl výběr možných datových typů. V rámci zachování jednoduchosti jsou dostupné pouze:

- Integer pro celá čísla
- Double pro desetinná čísla
- String pro řetězce
- Boolean pro logické hodnoty
- Char pro znakové hodnoty

Možným problémem může být různý rozsah stejných datových typů v různých jazycích, toho se nelze vyvarovat.

4.3 Ukázka

Na ukázce je vývojový diagram představující algoritmus počítání faktoriálu čísla. Zde je patrné odchýlení od normy u symbolu pro cyklus. Podle normy by měla spojnice ze symbolu, kde končí cyklus, vést do levého vstupu symbolu pro cyklus, a symboly, které už do cyklu nepatří, by měly být navázány spojnicí vycházející z pravé strany. Řešení neodpovídající normě bylo zvoleno z důvodu větší plynulosti diagramu v tomto případě. Je zde také vidět, že některé texty se nevešly do svých symbolů a bylo nutné rozhodnout, zda zvětšit všechny symboly a zachovat tak normu, nebo umožnit jejich selektivní zvětšování. Zmenšení velikosti písma nebylo vhodné z hlediska čitelnosti a je napevno dané v poměru k výšce symbolu.



Obrázek 16: Vývojový diagram, vlevo dle normy, vpravo s úpravou

5 Vlastní vývojové prostředí

5.1 Funkčnost

Prostředí vytvořené spolu s touto prací umožňuje:

- Tvorbu vývojových diagramů na pracovní ploše
- Specifikování konkrétní operace pro každý symbol
- Vytvoření proměnných nutných k práci
- Jednoduchou kontrolu kompletnosti a logické návaznosti diagramu
- Z vytvořeného diagramu generování zdrojového kódu v jazyce C#, Java a C
- Generování spustitelných programů za pomoci externího překladače jazyka C#
- Za pomoci speciálních proměnných generování programů pro desku Atmel AT89C51CC03 v jazyce C.
- Ze zdrojového kódu pro desku Atmel generování binárního souboru vhodného k nahrání do této desky za pomoci externího překladače SDCC
- Ukládání a opětovné načítání vytvořeného projektu
- Ukládání vygenerovaných kódů v textovém formátu
- Tisk diagramů i kódů

5.2 Implementace

Celé prostředí je vytvořeno v programu Microsoft Visual Studio 2008 v jazyce C# na platformě .NET 3.5. Pro běh nejsou nutné žádné externí knihovny, vyjma knihoven .NET Frameworku.

5.2.1 Hlavní třída programu

Třída reprezentující hlavní okno programu obsahuje metody pro základní ovládání většiny grafických komponent a jmenuje se jednoduše *Okno*. Její hlavní funkce jsou:

- Uchovávání seznamů symbolů, čar a proměnných aktuálního vývojového diagramu
- Uchovávání cest k použitým externím programům
- Kontrola změny diagramu kvůli nutnosti uložení
- Obsluha vykreslování pracovní plochy
- Ovládání grafických prvků
- Reakce na akce myší na pracovní ploše
- Metody pro ukládání a otevírání projektů
- Metody pro tisk
- Další pomocné metody

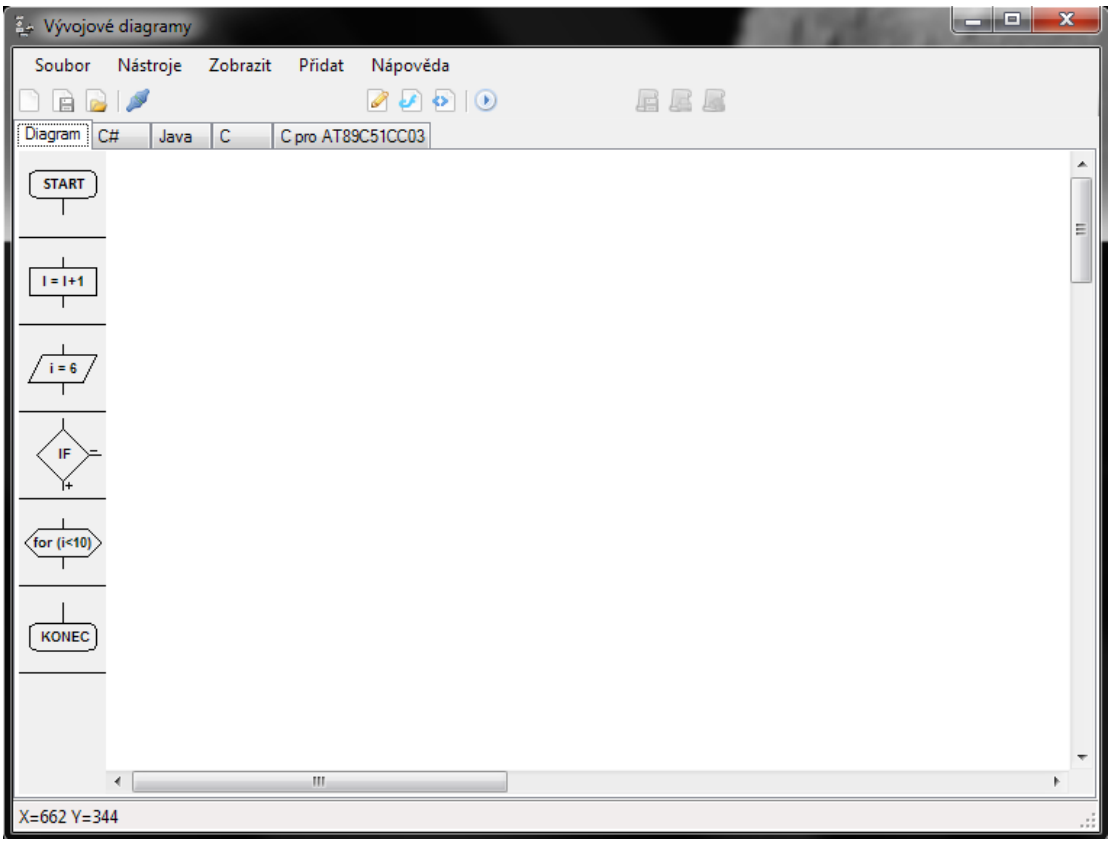
Grafické zpracování

Hlavní okno programu obsahuje většinu prvků nutných k tvorbě vývojového diagramu. V horní části je hlavní menu obsahující všechny dostupné funkce. Pod ním je lišta s ikonami pro nejpoužívanější funkce, což jsou:

- Nový projekt
- Uložit projekt
- Otevřít projekt
- Otevřít okno s cestami k externím programům
- Otevřít okno s poznámkami k projektu
- Otevřít okno proměnných
- Spustit kontrolu diagramu
- Spustit program

- Ikony pro práci se zdrojovým kódem (aktivním pouze v případě, že je zobrazen zdrojový kód)
    - Uložení kódu jako textového souboru
    - Tisk kódu
    - Generování binárního souboru (pouze při aktivaci poslední záložky)

Úplně dole je lišta zobrazující stavové informace a celému hlavním oknu dominuje oblast se záložkami pro diagram a jednotlivé jazyky. První záložka – diagram, obsahuje v levé části panel se symboly, které jdou přetahovat na pracovní plochu, která zabírá celou zbývající oblast. Další záložky obsahují pouze zdrojový kód.



Obrázek 17: Vývojové prostředí

5.2.2 Třída pracovní plochy

Pracovní plocha není nic jiného než grafická komponenta *Panel* s potlačenou metodou *onPaintBackground*. Velkým problémem, který byl nutně potřeba vyřešit, bylo problikávání bílé barvy při každém překreslování pracovní plochy. Překreslování se provádí při všech změnách pracovní plochy tj. při vložení nového symbolu, změně

obsahu nějakého symbolu, posunu pracovní plochy, ale hlavně při posouvání symbolů, nebo vytváření nové spojnice, kdy se překresluje několikrát za sekundu a problikávání kazí výsledný dojem. Problikávání je způsobeno tím, že při překreslování komponenty je nejdříve celá její plocha vyplněna barvou jejího pozadí a až poté jsou dokresleny přidané grafické prvky, v tomto případě symboly a spojnice. Toto chování se dá upravit již zmíněným potlačením metody *onPaintBackround* v kombinaci s použitím bufferu grafiky. Celá grafika se nejprve vykreslí do bufferu a potom se vzhled komponenty přepíše jeho obsahem. [4] Touto metodou minimalizujeme čas, kdy je viditelná část komponenty přepisována a je tedy v nekonzistentní podobě.

```
private void plocha_Paint(object sender, PaintEventArgs e)
{                                                          //g je grafika bufferu
    g.FillRectangle(Brushes.White, e.ClipRectangle);       //vyplnění pozadí bílou barvou
    Pen p = new Pen(Color.LightGray,1);                    //vytvoření pera pro okraje stránky
    p.DashStyle = DashStyle.Dash;                          //nastavení přerušované čáry pera
    g.DrawLine(p, new Point(800, 0),new Point(800,2228));  //vykreslení čar představujících konec
    g.DrawLine(p, new Point(0, 1114), new Point(1600, 1114));      //tisknutelné stránky
    foreach (Symbol s in seznam_symbolu) s.Vykresli(g,true,0,0);   //vykreslení dymbolů
    foreach (Cara c in seznam_car) c.Vykresli(g, true,0,0);    //vykreslení čar
    if (cara.Count > 1) g.DrawLines(new Pen(Color.Black, 2), cara.ToArray());  //vykreslení aktuální čáry
    buffer.Render(e.Graphics);                         //překreslení pracovní plochy obsahem bufferu
}
```

Ukázka kódu 1: Metoda vykreslující pracovní plochu

```
private Graphics g;                                        //grafika bufferu
private BufferedGraphicsContext currentContext;            //kontext bufferu
private BufferedGraphics buffer;                           //buffer

currentContext = BufferedGraphicsManager.Current;          //nastavení kontextu bufferu
buffer = currentContext.Allocate(plocha.CreateGraphics(), plocha.DisplayRectangle);  //vytvoření bufferu
g = buffer.Graphics;                                       //vytvoření grafiky bufferu k vykreslování
g.SmoothingMode = SmoothingMode.None;                      //nastaveni vyhlazovani
```

Ukázka kódu 2: Vytvoření bufferu grafiky

5.2.3 Třída „Symbol“

Bázová třída

Každý symbol diagramu je odvozen od abstraktní třídy, která uchovává vlastnosti a metody společné pro každý symbol.

Vlastnosti:

- Pozice na pracovní ploše
- Velikost

- Příznak vyplnění obsahu
- Příznak označení čáry
- Text popisku
- Barvu vykreslení
- Referenci na vstupní a výstupní čáry
- Pomocné proměnné pro pohyb symbolu a procházení diagramem

Metody:

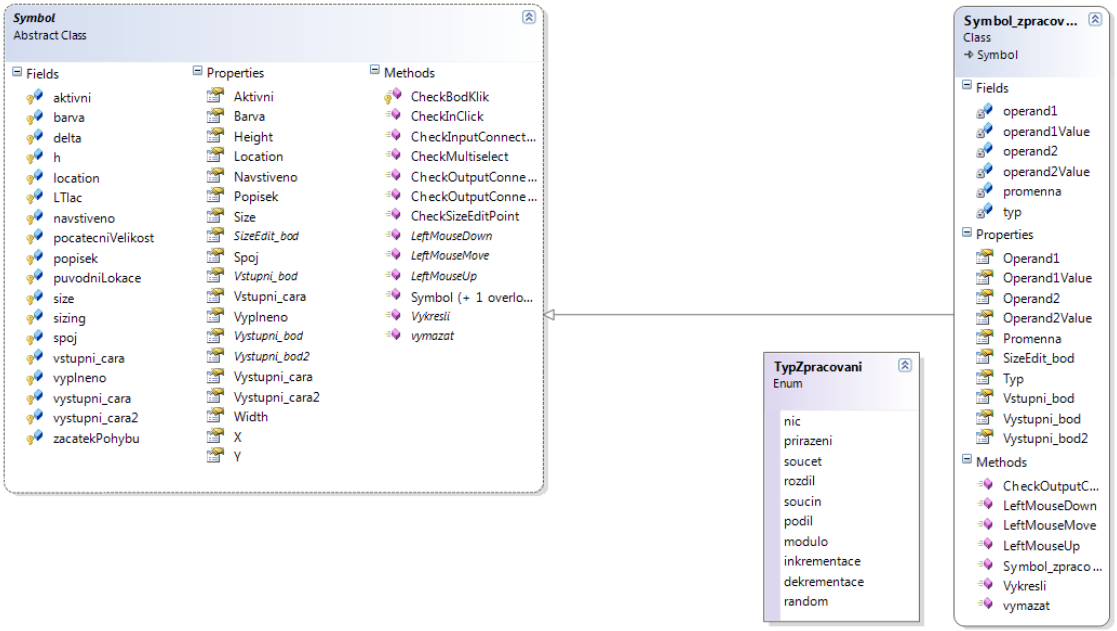
- Vykreslení symbolu
- Vymazání obsahu
- Kontrolu kliknutí na klíčové pozice v symbolu
- Definice chování při stisku, pohybu a uvolnění tlačítka myši

Třída konkrétního symbolu

Třída každého konkrétního symbolu je odvozena od bázové třídy, implementuje její abstraktní metody a přidává konkrétní funkce tvořeného symbolu. Jednotlivé symboly se od sebe liší v grafice, počtu vstupních a výstupních bodů a především v proměnných použitých pro generování kódu konkrétního programovacího jazyka.

```
private int promenna = -1;              //index do seznamu proměnných, -1 pokud je uložena přímá hodnota
public int Promenna { get { return promenna; } set { promenna= value; } }

private int hodnota = -1;               //index do seznamu proměnných, -1 pokud je uložena přímá hodnota
public int Hodnota { get { return hodnota; } set { hodnota = value; } }

private string hodnotaValue;            //index do seznamu proměnných, -1 pokud je uložena přímá hodnota
public string HodnotaValue { get { return hodnotaValue; } set { hodnotaValue = value; } }

private string operace;                 //operace porovnání
public string Operace { get { return operace; } set { operace = value; } }
```

Ukázka kódu 3: Vlastnosti pro generování kódu ze symbolu podmínky



Obrázek 18: Diagram tříd „Symbol“, „Symbol_zpracovani“ a výčtu „TypZpracovani“

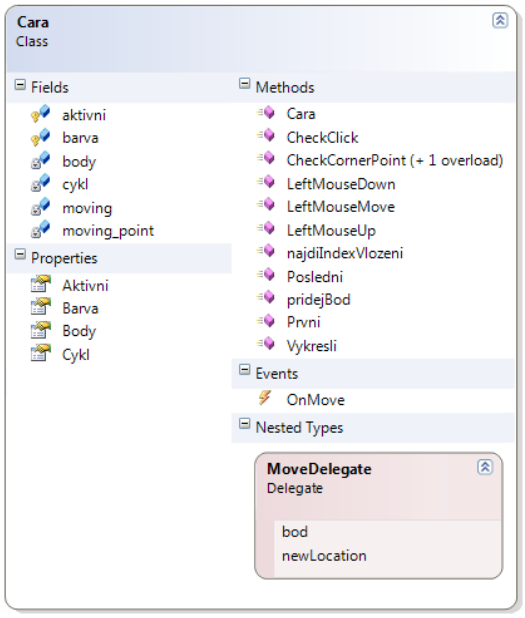
5.2.4 Třída „Cara“

Čára je reprezentována seznamem bodů, v němž vždy první leží ve výstupním bodě nějakého symbolu a poslední ve vstupním bodě jiného symbolu, nebo v bodě jiné čáry. Mimo seznamu bodů si čára uchovává:

- Barvu
- Příznak označení použitý při vykreslování
- Pomocné proměnné pro procházení grafu

Třída čáry obsahuje také tyto metody:

- Vykreslení
- Testování kliku myší s tolerancí 4px
- Testování kliku myší na rohové body
- Hledání indexu předchozího rohového bodu nutného pro vkládání nového bodu při spojování

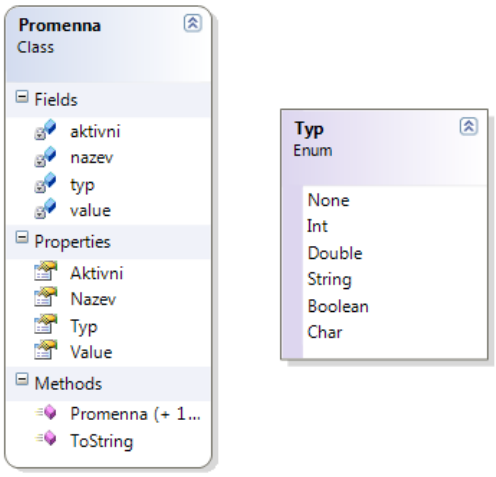


Obrázek 19: Diagram třídy „Cara“

5.2.5 Třída „Promenna“

Proměnná není nic jiného než soubor tří parametrů – názvu, datového typu a hodnoty. Mimo těchto tří hodnot obsahuje třída pro proměnnou ještě příznak aktivity. Příznak aktivity značí, jestli proměnná nebyla smazána, a je nutný, protože symboly si uchovávají odkaz na proměnnou ve formě indexu do seznamu proměnných, a když je nějaká proměnná smazána, musí se uchovat i nadále, kvůli zachování pořadí. Datový typ proměnné se určuje podle výčtu Typ.

Třída pro proměnnou obsahuje pouze jednu metodu a to přetíženou metodu *toString*, určenou pro textový výpis. Proměnné se implicitně vypisují ve tvaru „*jméno (datový_typ)*“.



Obrázek 20: Diagram třídy „Promenna“ a výčtu „Typ“

5.2.6 Třídy pro generování kódu

Bázová třída

Program umí generovat kód v jazycích C#, Java, C a C pro vývojovou desku. Každý z těchto jazyků je generován jiným generátorem, ale všechny jsou odvozeny od stejné bázové třídy *CodeGen*. Tato třída obsahuje metody společné pro všechny odvozené generátory, a to metodu kontrolující úplnost diagramu (vyplnění všech symbolů, propojení všech větví, zpětné skoky), pomocnou metodu *projdiDiagram*, která rekurzivně prochází diagramem od zvoleného symbolu až po symbol *KONEC*, dále metodu *najdiStart*, která ze seznamu vrátí symbol *START*, metodu najdi cíl, která vrátí symbol, do kterého vede zvolená čára, metodu *najdiRandom* která hlídá použití generování náhodného čísla v diagramu a metodu řádkuj, která doplní do vygenerovaného zdrojového kódu odřádkování.

Generátor konkrétního jazyka

Je jednou z nejdůležitějších tříd celého programu. Jejím úkolem je ze seznamu symbolů, čar, proměnných a textového komentáře vygenerovat zdrojový kód. Jedinou veřejnou metodou, která se dá zavolat, je metoda generuj a jejím parametrem je reference na komponentu, do které chceme kód vygenerovat. Generování probíhá ve více fázích. Nejdříve se spustí kontrola diagramu. V případě, že diagram není v pořádku, metoda vrátí *FALSE* a skončí. V opačném případě následuje vygenerování potřebných importů a hlavičky metody. Poté se zavolá metoda, která přidá do kódu proměnné podle seznamu a náležitě je inicializuje. Všechny proměnné jsou inicializovány na začátku metody. Poté se spustí rekurzivní procházení diagramu od symbolu *START* až po symbol *KONEC*. Procházení je jednoduchá operace, kdy se načte symbol, podle jeho typu a obsahu se vytvoří řádek kódu a stejná metoda se zavolá na symbol následující. Problém nastává, když aktuálně procházeným symbolem je symbol podmínky, který má 2 výstupy a vzniká tedy druhá větev kódu. V tomto případě se první spustí metoda, která najde symbol, před nímž se větvení ukončuje. Poté se projde první větev až po tento symbol, následně druhá až po spojení a dále se už pokračuje klasicky jednou větví. Podobný princip je použit u symbolu cyklu, kde ovšem není nutné procházet druhou větví. Důležité je pouze místo, kde druhá větev končí, protože tam končí i celý cyklus. Takto se prochází celý diagram a rekurze se ukončí se symbolem *KONEC*. Po ukončení procházení diagramu se přidají pouze ukončení hlavní metody, popřípadě hlavní třídy, a úplně nakonec se kód upraví odřádkováním.

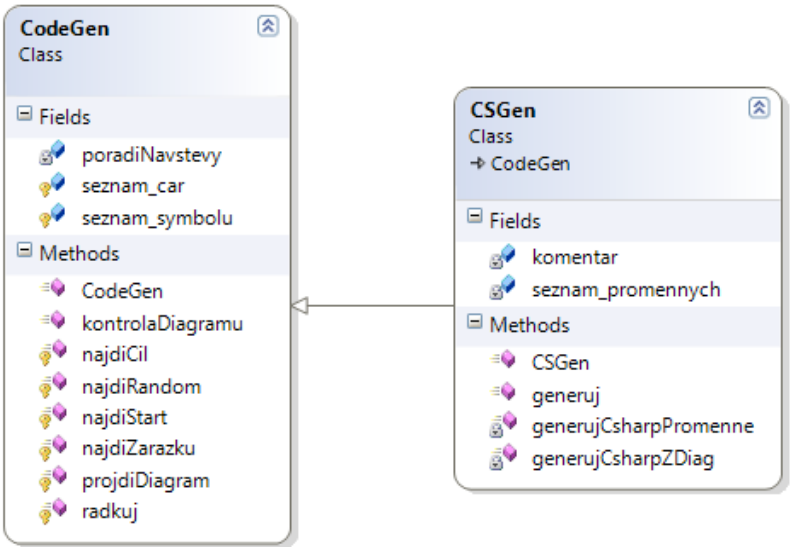
```
public bool generuj(RichTextBox box)                    //parametrem je RichTextBox do kterého se generuje
{                                                       //metoda vrací false pokud generování selhalo
if (!kontrolaDiagramu()) return false;                          //volání kontroly diagramu
    StringBuilder text = new StringBuilder();                   //vytvoření string builderu pro kód
    text.AppendLine("/*");
    text.AppendLine(komentar);                                  //připojení komentáře projektu
    text.AppendLine("*/\n\n");
    text.AppendLine("using System;");                          //definice jmenného prostoru, třídy a metody
    text.AppendLine();
    text.AppendLine("namespace App");
    text.AppendLine("{");
    text.AppendLine("class App");
    text.AppendLine("{");
    text.AppendLine("static void Main(string[] args)");
    text.AppendLine("{");

    //volání metody, která vygeneruje deklarace proměnných
    generujCsharpPromenne(text);
    //pokud je použito generování náhodného čísla, přidá se řádek pro vytvoření objektu random
    if (najdiRandom(najdiStart())) text.AppendLine("Random random = new Random();");
    text.AppendLine();
    //pokud jdou v diagramu nějaké symboly, spustí se generování
    if (seznam_symbolu.Length > 0) generujCsharpZDiag(null, text, null);

    //ukončení bloku, metody, třídy a namespace
    text.AppendLine("}");
    text.AppendLine("}");
    text.AppendLine("}");
    box.Text = text.ToString();
    //kód se odřádkuje podle znaků „{„ a „}“
    radkuj(box);
    //generování proběhlo vpořátku
    return true;
}
```

Ukázka kódu 4: Metoda generuj ve třídě „CSGen“

```
//parametry jsou aktuální symbol, StringBuilder s kódem a zarážka použitá při procházení více větví
private void generujCsharpZDiag(Symbol s, StringBuilder text, Symbol zarazka)
{
    if (s == null)
        generujCsharpZDiag(najdiStart(), text, null);    //první volání metody, začíná symbolem START
    if (s == zarazka)  return;                                  //konec větve
    if (s is Symbol_konec)                                      //aktuální symbol je symbol KONEC,
    {                                                           //připojení řádků s jeho obsahem a ukončení
        Symbol_konec sk = s as Symbol_konec;
        if (sk.Vypsat)
            text.AppendLine("\nConsole.WriteLine(\"" + sk.Text + "\");");
        if (sk.Cekat)
            text.AppendLine("Console.ReadLine();");
        return;
    }
    if (s is Symbol_start)
    {                                                                   //aktuální symbol je symbol START,
       Symbol_start ss = s as Symbol_start;                                 //připojení řádku s jeho obsahem
       if (ss.Vypsat)
```

```
            text.AppendLine("Console.WriteLine(\"" + ss.Text + "\");\n");
        generujCsharpZDiag(najdiCil(s.Vystupni_cara), text, null);       //rekurzivní pokračování
    }                                                                    //následujícím symbolem
    if (s is Symbol_vstvyst)            //aktuální symbol je symbol vstup/výstup
    {
        …
        //připojení řádku s obsahem symbolu
        …
        generujCsharpZDiag(najdiCil(s.Vystupni_cara), text, zarazka);
    }
    if (s is Symbol_zpracovani)         //aktuální symbol je symbol zpracování
    {
        …
        //připojení řádku s obsahem symbolu
        …
        generujCsharpZDiag(najdiCil(s.Vystupni_cara), text, zarazka);
    }
    if (s is Symbol_cyklu)              //aktuální symbol je symbol cyklu
    {
        …
        //připojení řádku s obsahem symbolu
        …
        //zde je použita zarážka, zarážkou je symbol, u kterého končí druhá větev. Generování první větve
        //se ukončí, když je aktuální symbol zarážkou
        generujCsharpZDiag(najdiCil(s.Vystupni_cara), text, najdiCil(s.Vystupni_cara2));
        text.AppendLine("}");
        //pokračování druhou větví
        generujCsharpZDiag(najdiCil(s.Vystupni_cara2), text, zarazka);
    }
    if (s is Symbol_podminky)           //aktuální symbol je symbol cyklu
    {
        //symbol podmínky je generován obdobně jako symbol cyklu
```

Ukázka kódu 5: Zjednodušená metoda pro generování kódu z konkrétních symbolů



Obrázek 21: Diagram tříd „CodeGen“ a „CSGen“

5.2.7 Ukládání a načítání projektu

Program umožňuje vytvořené projekty ukládat a posléze načítat ve vlastním formátu **.vvd*. Tento formát není ničím jiným, než souborem serializovaných objektů. Konkrétně se skládá z:

- Počtu uložených symbolů
- Počtu uložených čar
- Počtu uložených proměnných
- Uložených symbolů
- Uložených čar
- Uložených proměnných
- Textu komentáře programu

Při ukládání bylo vhodné docílit co nejmenší velikosti souborů. Toho bylo dosaženo dvěma způsoby. Prvním je upravení průběhu serializace jednotlivých objektů pomocí atributu *[NonSerialized]* přidaného k proměnným, které není nutné ukládat. Těmito proměnnými jsou například pomocné proměnné pro procházení diagramu u symbolů, nebo příznak aktivity (označení) čáry nebo symbolu. Hodnoty proměnných, které jsou označené jako *NonSerialized* nejsou při serializaci ukládány do proudu a při deserializaci jsou nastaveny na výchozí hodnoty. [5] Druhým způsobem zmenšení velikosti výsledného souboru je použití proudové komprese *GZip*. Komprese zmenší výsledný soubor přibližně na polovinu. Kombinací těchto dvou metod se výsledná velikost středně složitého projektu pohybuje řádově v desítkách kB.

Problémem, který se při ukládání projektů pomocí serializace objevil, byl rozpad vazeb mezi symboly a čárami. Každý symbol si udržuje odkaz na čáry, které k němu patří, a tyto odkazy nejsou při serializaci korektně obnoveny. Řešením je funkce, která pro každý vstupní nebo výstupní bod každého symbolu hledá čáru, která v tomto bodě končí nebo začíná a poté vytvoří korektní provázání.

Alternativou, která není implementována, by mohlo být ukládání jednotlivých objektů a jejich proměnných v textové podobě. Výhodou tohoto přístupu je jednoduchá čitelnost souborů pro člověka a případná přenositelnost mezi platformami, popřípadě různými verzemi programu. Nevýhodou je velikost výsledných souborů, případně způsobení nekonzistence dat neodborným zásahem. Určitým kompromisem by mohla

být serializace v nějakém značkovacím jazyce, například *xml*. To by výslednou strukturu zpřehlednilo, ale zmíněné nevýhody by přetrvávaly.

5.2.8 Generování spustitelného programu

Platforma PC

Vývojové prostředí umožňuje po sestavení úplného vývojového diagramu a následného vygenerování kódu vytvořit spustitelný program pro platformu PC. Bylo potřeba zvolit jazyk, z kterého se bude program generovat. V úvahu připadal C#, Java a C. Nakonec byl zvolen jazyk C#, protože celé prostředí je v něm napsáno a aby mohlo fungovat, je nutné mít nainstalován .NET Framework, jehož součástí je i kompilátor. To znamená, že pokud půjde prostředí spustit, půjde i vygenerovat spustitelný program, bez nutnosti instalovat externí kompilátor. Samotné generování probíhá v několika krocích:

1) Kontrola diagramu pomocí třídy *CodeGen*
2) Uložení projektu
3) Generování zdrojového kódu v jazyce C#
4) Uložení zdrojového kódu
5) Zavolání překladače C#
6) Spuštění vygenerovaného programu, nebo výpis chyb kompilátoru

Vývojová deska

Druhou platformou, pro níž je možné vygenerovat program je vývojová deska Atmel AT89C51CC0 s procesorem 8051. Programy pro tuto desku je možné psát v jazyce assembler nebo C. V tomto případě byl zvolen jazyk C. Aby byl program jednoduchý a šly ovládat periferie, které deska obsahuje, je nutné použít externí soubory obsahující metody pro práci s těmito periferiemi. Tyto metody už bude tvořený program pouze volat. Soubor, který tyto metody obsahuje, se jmenuje *pripravek.inc* a je ve složce data, v kořenové složce programu. Periferiemi, jejichž ovládání je v tomto souboru obsaženo, jsou klávesnice, *LedBar*, LCD displej a sériová linka. Vyjma metod pro ovládání periferií obsahuje tento soubor i metodu *PerifInit*, která nastaví potřebná přerušení, vynuluje *LedBar*, vynuluje LCD a nastaví implicitní výstup na displej. Aby se daly jednoduše volat metody pro periferie a ovládat porty procesoru pomocí

vývojového diagramu, je nutné vytvořit speciální proměnné pomocí „menu-přidat-Proměnné pro přípravek AT89C51CC0“, konkrétní funkce všech proměnných jsou popsány v nápovědě.

Mimo tohoto souboru je pro generování nutný ještě hlavičkový soubor *AT89C51CC03.h*, který obsahuje definice jmen registrů procesoru a soubor *TYPY.H,* který obsahuje definice datových typů.

Samotnou kompilaci vygenerovaného zdrojového kódu do binárního souboru má na starosti externí program SDCC (Small device C compiler). Aby tento program mohl být volán, musí být v počítači nainstalován a v prostředí musí být vyplněna cesta k některým jeho souborům.

Generování binárního souboru, který lze nahrát do procesoru, probíhá v těchto krocích:

1) Kontrola diagramu
2) Generování kódu v jazyce C
3) Kontrola vyplnění cest k překladači SDCC
4) Uložení projektu
5) Uložení zdrojového kódu
6) Zkopírování souborů se zdrojovými kódy pro procesor a periferie do složky s projektem
7) Zavolání překladače SDCC
8) Kontrola chyb překladače
9) Vytvoření dávkového souboru, který převede vytvořený Intel binární soubor (přípona *ihx*) na klasický binární soubor (přípona *hex*)
10) Spuštění dávkového souboru
11) Smazání nepotřebných souborů, které vytvořil překladač a souborů pro periferie

Na konci tohoto procesu ve složce projektu zůstane soubor *název_projektu.hex,* který je možné nahrát do vývojové desky nebo jejího simulátoru a soubor *název_projektu.rst*, který obsahuje ladící informace.

6 Závěr

V první části práce jsou uvedeny příklady známých i méně známých vizuálních jazyků, používaných v širokém spektru odvětví. Počínaje u průmyslu a konče programováním hraček, které zvládnou i děti. Ke každému jazyku je uveden příklad prostředí, ve kterém ho lze použít, jsou popsány základní vlastnosti těchto prostředí a komfort programování v nich.

V druhé části jsou popsány vývojové diagramy dle normy ČSN ISO 5807 tak, jak jsou použity ve vývojovém prostředí, které je součástí této práce. Mimo samotných vývojových diagramů jsou uvedeny principy tvorby zdrojového kódu z vývojového diagramu s potřebnými náležitostmi.

V poslední části je rozebrána implementace vývojového prostředí, které s touto prací vzniklo a dle zadání umožňuje základní práci s vývojovými diagramy a jejich převod na zdrojové kódy v jazyce C#, Java a C a následnou kompilaci. Dále jsou popsány hlavní třídy programu a zdrojové kódy důležitých metod.

Samotné prostředí může najít své uplatnění hlavně při výuce algoritmizace a hledání souvislostí mezi vývojovým diagramem a zdrojovým kódem. Každý symbol vývojového diagramu zastává právě jednu instrukci, což umocňuje vztah mezi symboly a jednotlivými řádky kódu. Další funkcí, která není až tak běžná v dostupném software, je porovnání stejných algoritmů v různých programovacích jazycích. Představě o výsledné funkci algoritmu pomáhá i možnost vygenerovat spustitelný program, popřípadě program ovládající hardwarové periferie vývojové desky.

Prostředí poskytuje prostor pro velké množství rozšíření a úprav. Jednou z možností je implementace debuggeru vytvořených programů. Provizorní možností je kompilace programu s parametrem *„-debug+“*, kdy se vytvoří mimo spustitelný program i soubor debuggovacích symbolů a je možné program testovat v externím debuggeru pro jazyk C#. Dalším nabízejícím se rozšířením je přidání dalších programovacích jazyků.

Použitá literatura

[1] JOHN, Karl-Heinz a Michael TIEGELKAMP. IEC 61131-3: Programming industrial automation systems: *concepts and programming languages, requirements for programming systems, aids to decision-making tools*. New York: Springer, c2001, 376 s. ISBN 35-406-7752-6.

[2] ČSN ISO 5807. *Zpracování informací*: *DOKUMENTAČNÍ SYMBOLY A KONVENCE PRO VÝVOJOVÉ DIAGRAMY TOKU DAT, PROGRAMU A SYSTÉMU, SÍŤOVÉ DIAGRAMY PROGRAMU A DIAGRAMY ZDROJŮ SYSTÉMU*. První vydání. Praha: ČESKÝ NORMALIZAČNÍ INSTITUT, 1996.

[3] Picaxe. Www.hobbyrobot.cz [online]. 03.07.2006, 28.07.2011 [cit. 2012-04-07]. Dostupné z: http://www.hobbyrobot.cz/picaxe.htm

[4] SELLS, Chris. *C# a Winforms*: *programování formulářů windows*. Vyd. 1. Brno: Zoner Press, 2005, 648 s. ISBN 80-868-1525-0.

[5] TROELSEN, Andrew. *C# a .NET 2.0 profesionálně*. Vyd. 1. Brno: Zoner Press, 2006, 1197 s. ISBN 80-868-1542-0.

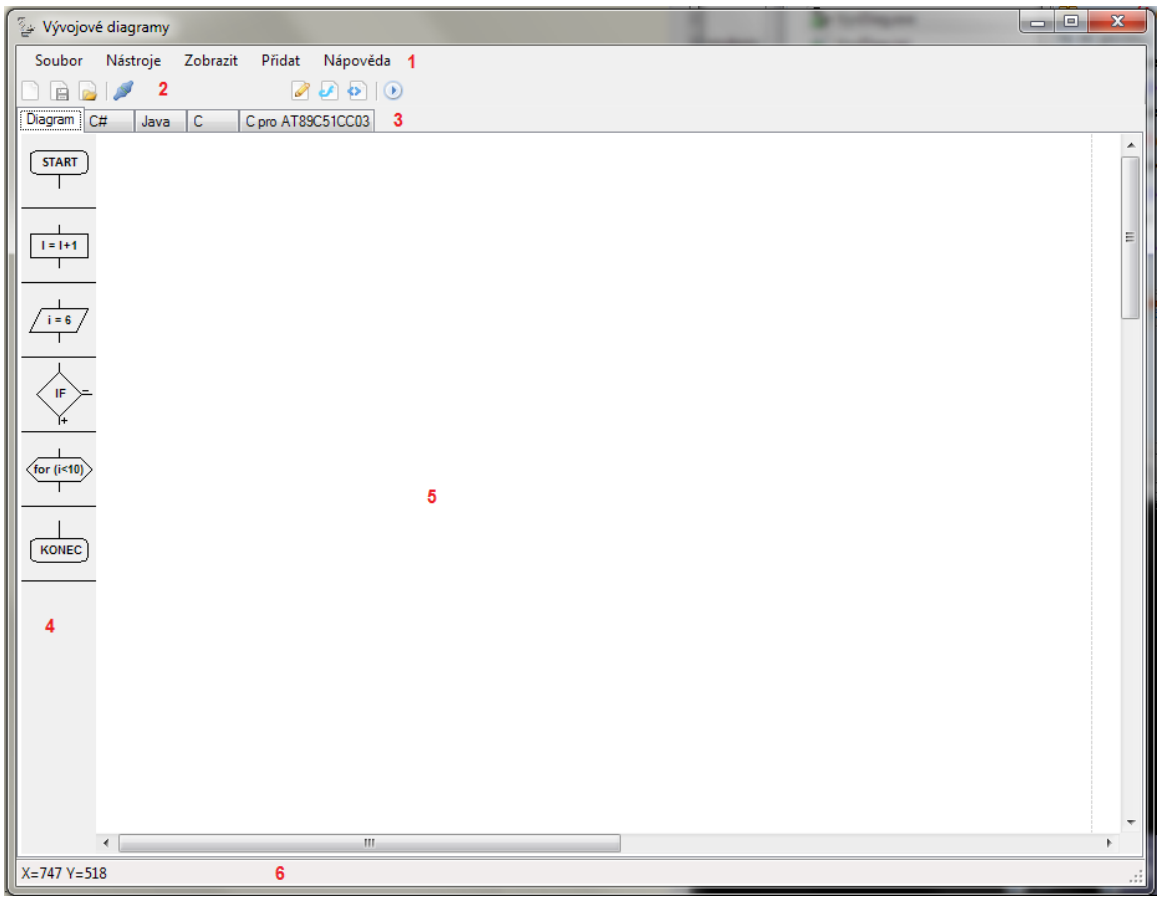
Příloha A: Návod k obsluze

1 Popis prostředí

1.1 Hlavní menu

Soubor	
- Nový	Založení nového projektu
- Otevřít	Otevřít projekt ze souboru
- Uložit	Uložit projekt
- Uložit jako	Uložit projekt s upřesněním jména
- Náhled tisku	Otevřít okno s náhledem tisku
- Tisk	
- Konec	Ukončit program
Nástroje	
- Kontrola diagramu	Spustí kontrolu vytvořeného diagramu.
- Spustit program	Z diagramu vytvoří **.exe* soubor a spustí ho
Zobrazit	
- Okno proměnných	Zobrazí okno pro práci s proměnnými
- Cesty	Zobrazí okno pro upřesnění cest ke komponentám
- Popis projektu	Zobrazí okno pro vyplnění popisu projektu
Přidat	
- Proměnnou	
- Symbol	
- Proměnné pro AT89C51CC03	Přidá proměnné pro ovládání přípravku
Nápověda	
- Nápověda	
- Seznam funkčních kláves	
- O programu	

1.2 Hlavní okno programu



Obrázek 1: Hlavní okno programu

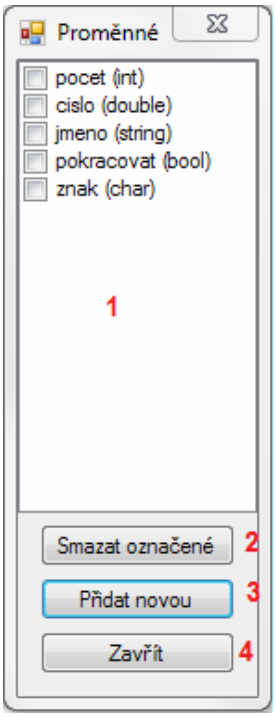
1) Menu
2) Lišta s rychlým přístupem
   a) Nový projekt
   b) Uložit
   c) Otevřít
   d) Zobrazit okno cest
   e) Zobrazit okno popisu projektu
   f) Zobrazit okno proměnných
   g) Spustit kontrolu diagramu
   h) Vygenerovat *.exe* soubor a pustit
   i) Uložit kód
   j) Vytisknout kód

k) Generovat *.hex* soubor

3) Záložky pro diagram a jednotlivé programovací jazyky
4) Panel se symboly
5) Hlavní plocha
6) Stavová lišta

1.3 Okno proměnných

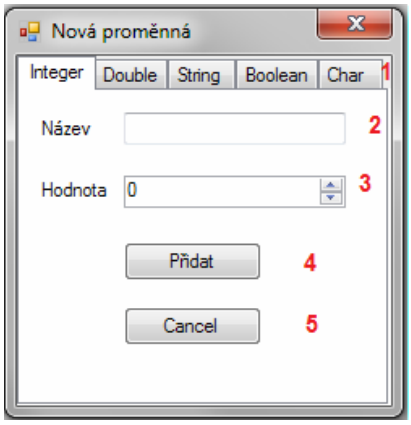
Okno proměnných slouží k vytváření nebo mazání proměnných. Toto okno lze vyvolat z menu, ikonou v hlavním okně, nebo tlačítkem s „+“ ve vlastnostech každého symbolu.



Obrázek 2: Okno proměnných

1) Seznam vytvořených proměnných
2) Smaže označené proměnné
3) Přidá novou proměnnou
4) Zavře okno

1.4 Okno přidání nové proměnné

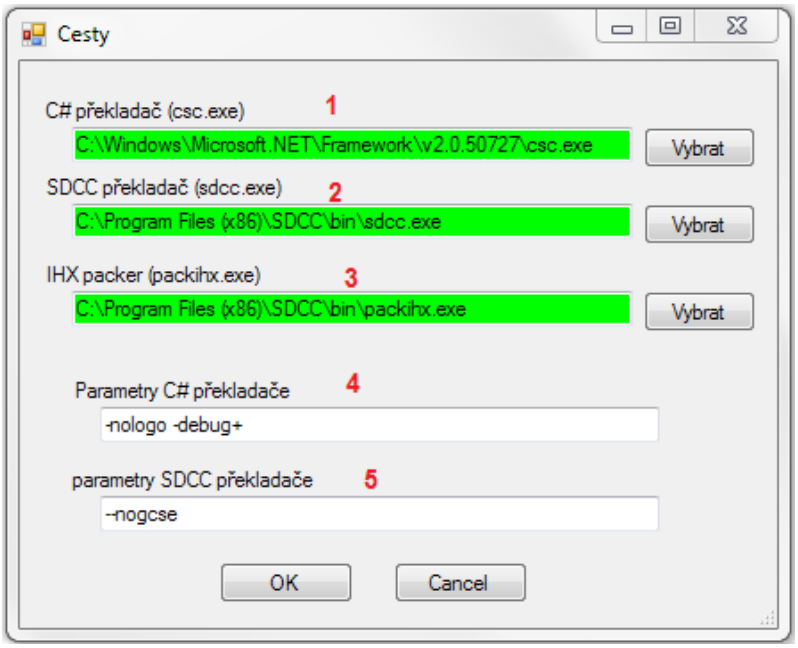


Obrázek 3: Okno pro přidání nové proměnné

1) Záložky pro různé datové typy
2) Název nové proměnné
3) Počáteční hodnota
4) Přidat
5) Zavřít okno

1.4 Okno cest

Okno cest složí k nastavení cest k překladačům a nastavení jejich parametrů. Okno lze vyvolat z menu nebo ikonou v hlavním okně.



Obrázek 4: Okno cest

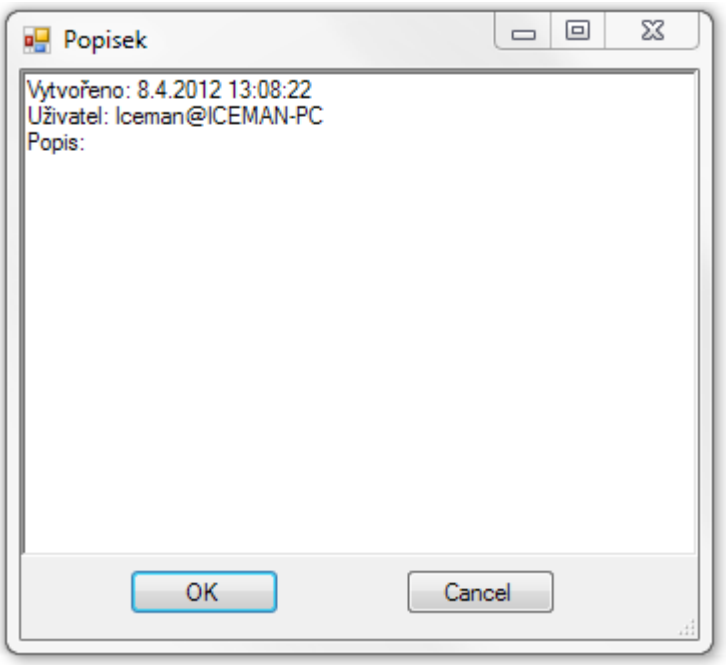
1) Cesta ke kompilátoru jazyka C#. Tento kompilátor je dodávaný k .NET Frameworku, který je nutný pro běh programu a tudíž byl zvolen jako kompilátor pro výsledné soubory s příponou *exe*.
2) Cesta ke kompilátoru jazyka C. Nástroj SDCC je použit ke generování souboru s příponou *ihx* vhodného k nahrání do desky.
3) Cesta k IHX *packeru*. IHX *packer* převede intel *hex* soubor (*ihx*) na klasický binární soubor a je součástí balíku SDCC.
4) Parametry C# kompilátoru. Implicitně *„-nologo*“ a *„–debug+“*. První parametr zabraňuje zobrazení loga při překladu a druhý nutí kompilátor vytvořit symboly pro testování.
5) Parametry SDCC kompilátoru. Implicitně *„—nogcse*“. Tento parametr slouží pro upravení optimalizace.

Pokud po vybrání cesty soubor neexistuje nebo je špatný, políčko zčervená. V opačném případě je zelené.

Open source nástroj SDCC je možné získat na adrese: http://sdcc.sourceforge.net/

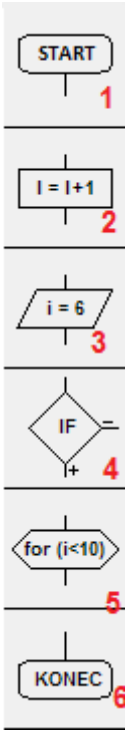
1.5 Okno popisu projektu

Ke každému projektu se vygeneruje popisek, který je možné doplnit v tomto okně, text popisku se automaticky přidá na začátek každého zdrojového kódu. Okno lze vyvolat z menu nebo ikonou v hlavním okně.



Obrázek 5: Okno popisu projektu

2 Dostupné symboly

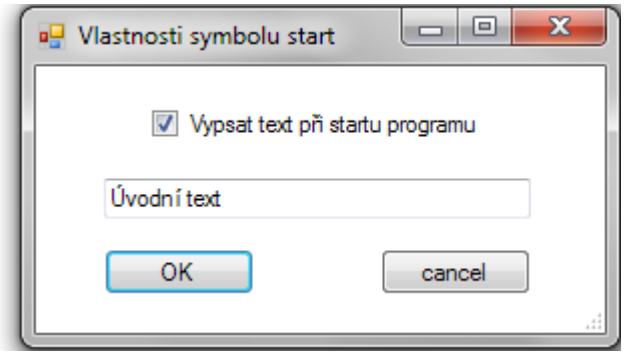


Obrázek 6: Dostupné symboly

1) Symbol start
2) Symbol zpracování dat
3) Symbol pro vstup a výstup
4) Symbol podmínky
5) Symbol cyklu
6) Symbol konec

2.1 Symbol START

Každý program musí začínat symbolem *START*. Jeho jedinou možnou funkcí je vypsat při začátku programu nějaký text

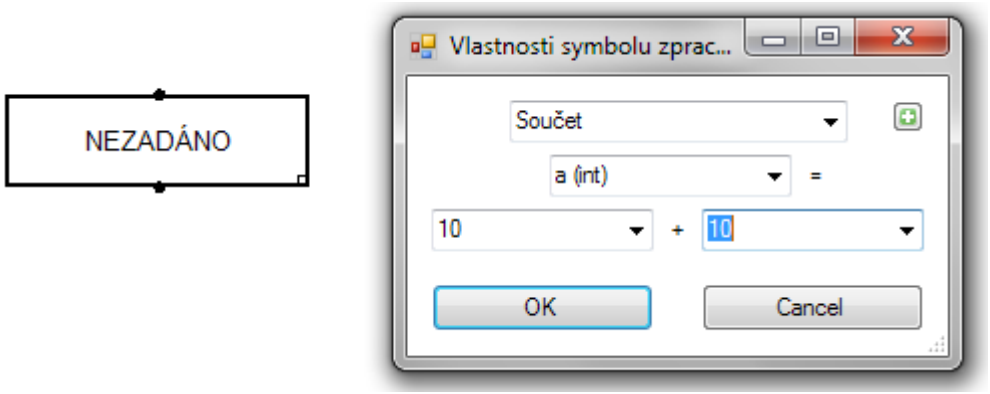


Obrázek 7: Symbol START

2.2 Symbol ZPRACOVÁNÍ

Tento symbol slouží pro operace s daty nebo proměnnými. Zatím implementované funkce jsou přiřazení, součet, rozdíl, podíl, dělení modulo, součin, inkrementace, dekrementace a generování náhodného čísla.

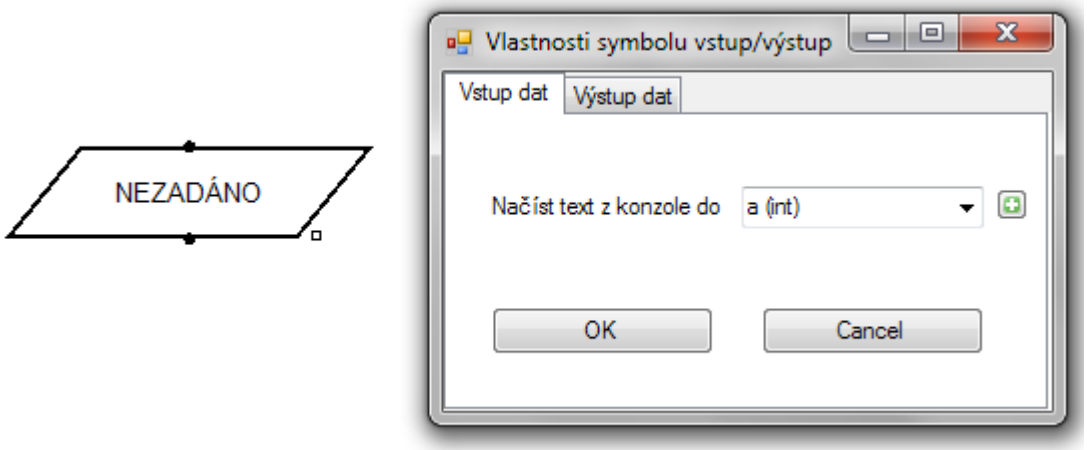
Prvním parametrem je vždy proměnná, do níž chceme výsledek uložit a poté následují jedna, nebo dvě proměnné, nebo hodnoty.



Obrázek 8: Symbol ZPRACOVÁNÍ

2.3 Symbol VSTUPU a VÝSTUPU

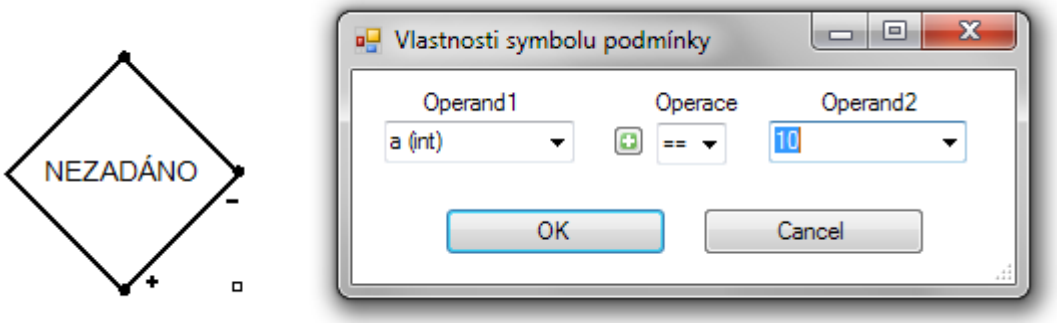
Tento symbol má dvě funkce, načítání dat z klávesnice nebo výpis dat na konzoli. Obě varianty mají jen jeden parametr, tj. proměnnou pro uložení vstupu, nebo výpis, nebo text pro výpis.



Obrázek 9: Symbol VSTUPU a VÝSTUPU

2.4 Symbol PODMÍNKY

Tento symbol slouží k větvení programu. Obsahuje dva parametry (proměnné nebo hodnoty) a symbol pro jejich porovnání. Výsledkem porovnání je logická hodnota pravda/nepravda. Pokud je výsledkem pravda, program pokračuje výstupem dole, pokud opak, program pokračuje výstupem vpravo.

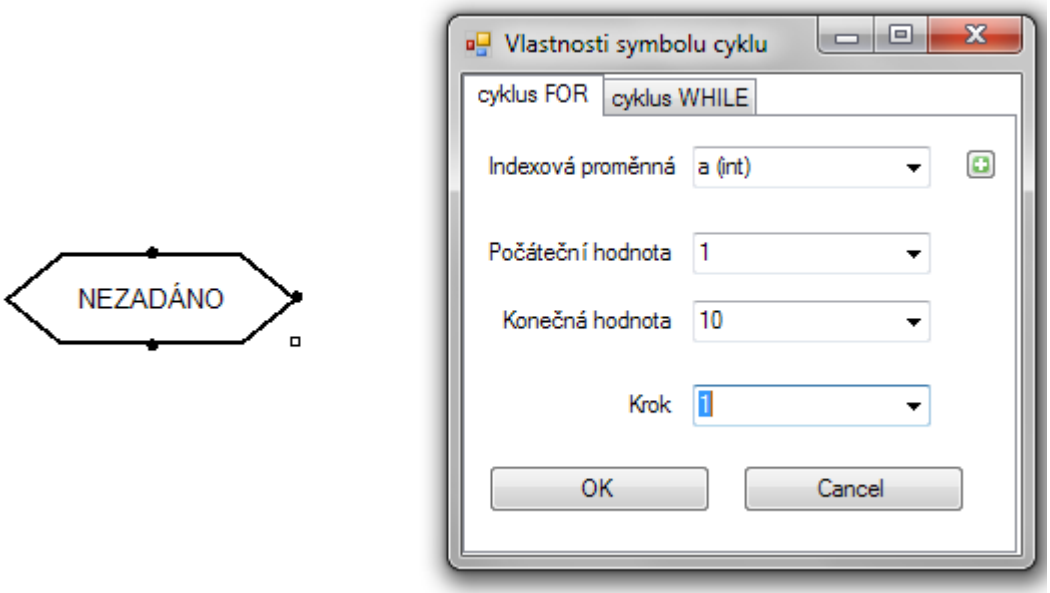


Obrázek 10: Symbol PODMÍNKY

2.5 Symbol CYKLU

Tento symbol slouží k vícenásobnému opakování nějakých příkazů. Vyskytuje se ve dvou formách, s pevným počtem opakování (*FOR*) a s podmínkou (*WHILE*).

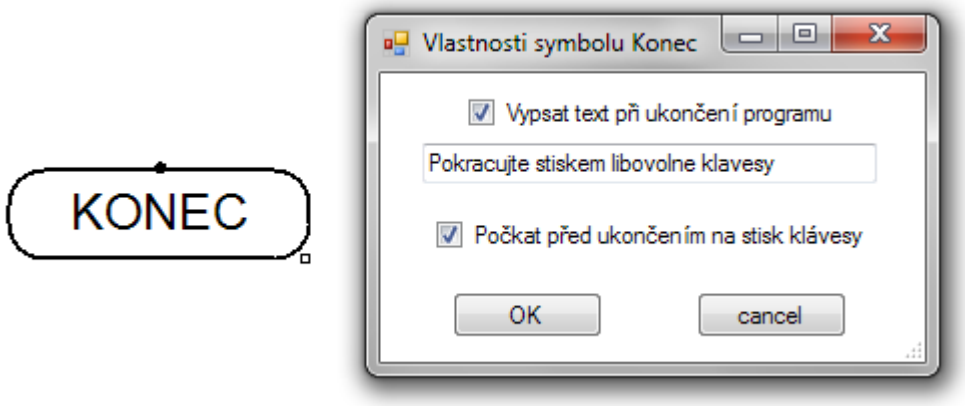
Cyklus *FOR* má 4 parametry, indexovou proměnnou, ve které se mění index průchodu, počáteční hodnotu, krok a konečnou hodnotu. Cyklus *WHILE* má na svém začátku podmínku a probíhá, dokud je tato podmínka splněna.



Obrázek 11: Symbol CYKLU

2.6 Symbol KONEC

Každý program musí být ukončen tímto symbolem. Implementovány jsou dvě funkce, výpis textu na konci programu a možnost ukončení až po stisku klávesy.



Obrázek 12: Symbol KONEC

3 Datové typy a proměnné

Integer

Celé číslo v rozsahu podle zvoleného jazyka.

Double

Desetinné číslo v rozsahu a přesnosti podle jazyka. Jako oddělovač desetinných míst se používá desetinná čárka.

String

Řetězec znaků. Každý řetězec musí být z obou stran ohraničen uvozovkami (").

Boolean

Logická hodnota. Jediné přípustné hodnoty jsou *true* nebo *false*.

Char

Jeden znak. Hodnotou musí být pouze jeden znak, z obou stran ohraničený apostrofy (').

4 Ukázka tvorby programu

4.1 Vyšší programovací jazyk

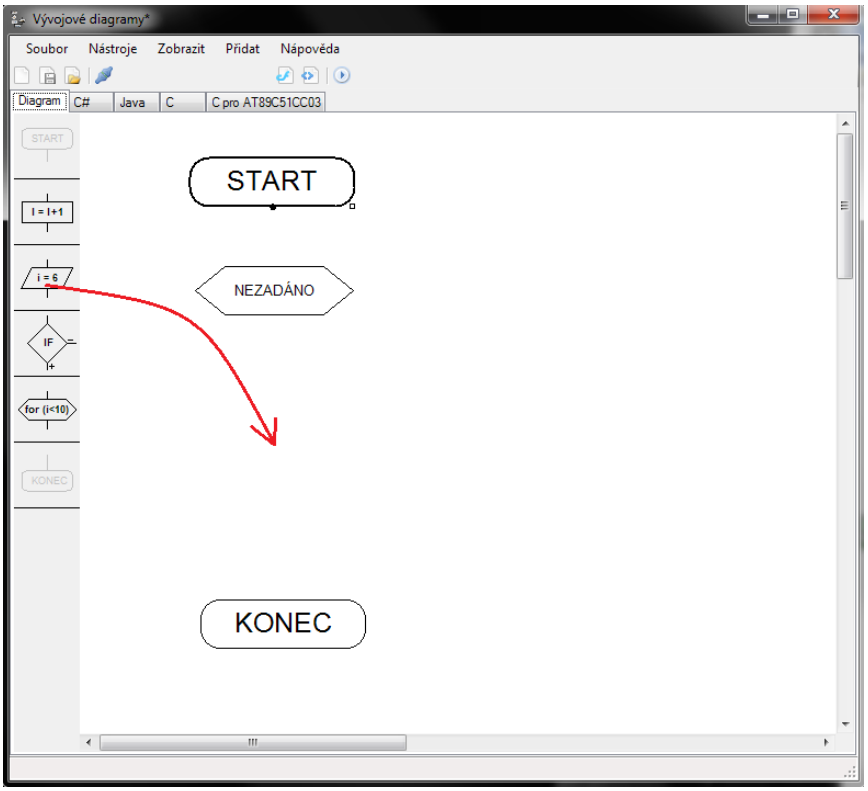
Vytvoření programu pro výpis čísel od 1 do 10.

Krok 1.

Založení nového projektu (např. Soubor->Nový), kontrola cest k překladačům. V našem případě je nutný jen překladač C# (*csc.exe*). Jeho cesta by měla být v pořádku i bez nastavování. Implicitně je nastavena na překladač .NET Framework 2.0

Krok 2.

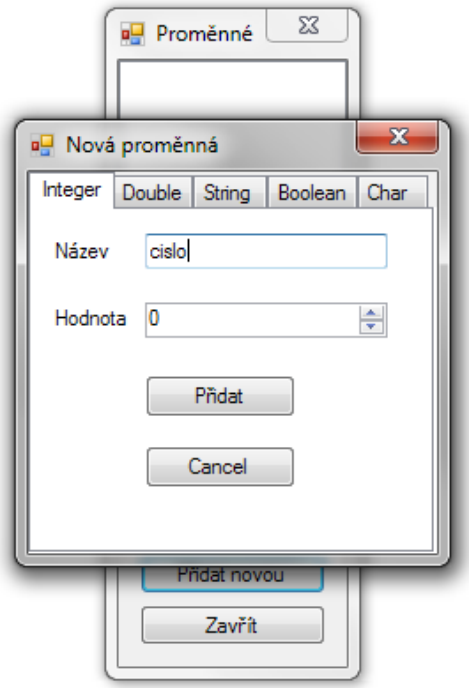
Přetáhnutí potřebných symbolů z panelu na pracovní plochu. Symboly se přetahují metodou *drag and drop*. Povinné symboly pro každý program jsou *START* a *KONEC*, dále bude nutný symbol pro cyklus s pevným počtem opakování (*FOR*) a symbol pro výpis.



Obrázek 13: Vytvoření symbolů

Krok 3.

Vytvoření proměnných. Pro tuto jednoduchou ukázku budeme potřebovat pouze jednu proměnnou a to číslo, které budeme vypisovat. Otevřeme okno proměnných a vytvoříme proměnnou datového typu integer (celé číslo) s názvem *cislo*. Její počáteční hodnota může být jakákoliv, protože cyklus *FOR* ji bude nastavovat za nás.



Obrázek 14: Vytvoření proměnné

Krok 4.

Vyplnění symbolů. Teď musíme vytvořené symboly vyplnit. Pokud máme označený symbol a stiskneme klávesu *enter* nebo na nějaký symbol dvakrát klikneme, zobrazíme okno s jeho vlastnostmi.

4a.

Ve vlastnostech symbolu start zaškrtneme políčko „Vypsat text při startu“ a dopíšeme například „Budou vypsány čísla 1-10“.

4b.

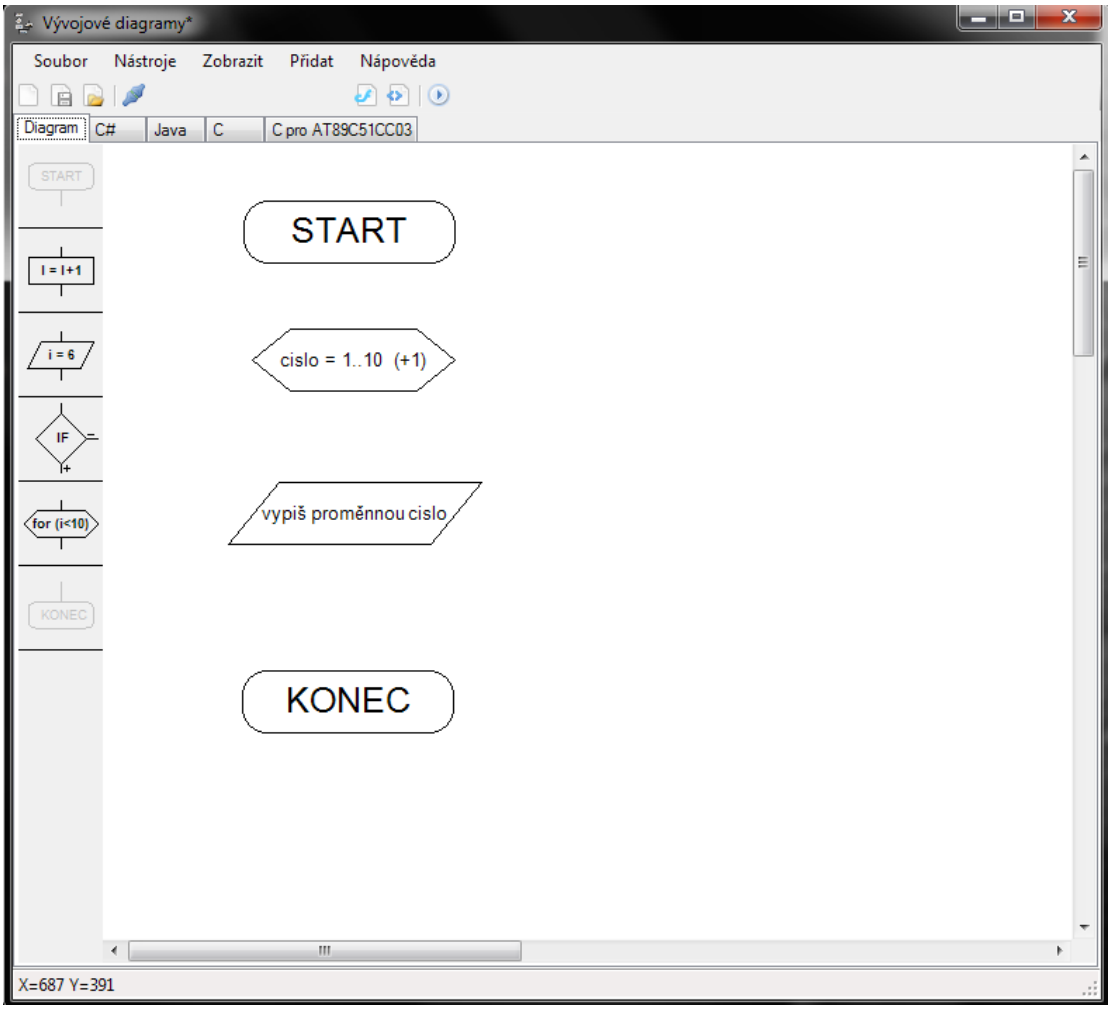
Ve vlastnostech symbolu cyklu se přesuneme na záložku „cyklus FOR“. Jako indexovou proměnnou vybereme naši proměnnou *cislo*. Jako počáteční hodnotu napíšeme 1, jako konečnou 10 a krok 1.

4c.

V symbolu pro výpis se přepneme na záložku „Výstup dat“ a vybereme naši jedinou proměnnou *cislo*.

4d.

Pokud bychom už dále nic nevyplňovali, program by fungoval, ale po vypsání čísel by se ukončil tak rychle, že bychom neměli možnost postřehnout výsledek. Proto ve vlastnostech symbolu *KONEC* zaškrtneme obě políčka a dopíšeme text „Pokračujte stiskem libovolné klávesy“.



Obrázek 15: Vyplněné symboly

Krok 5.

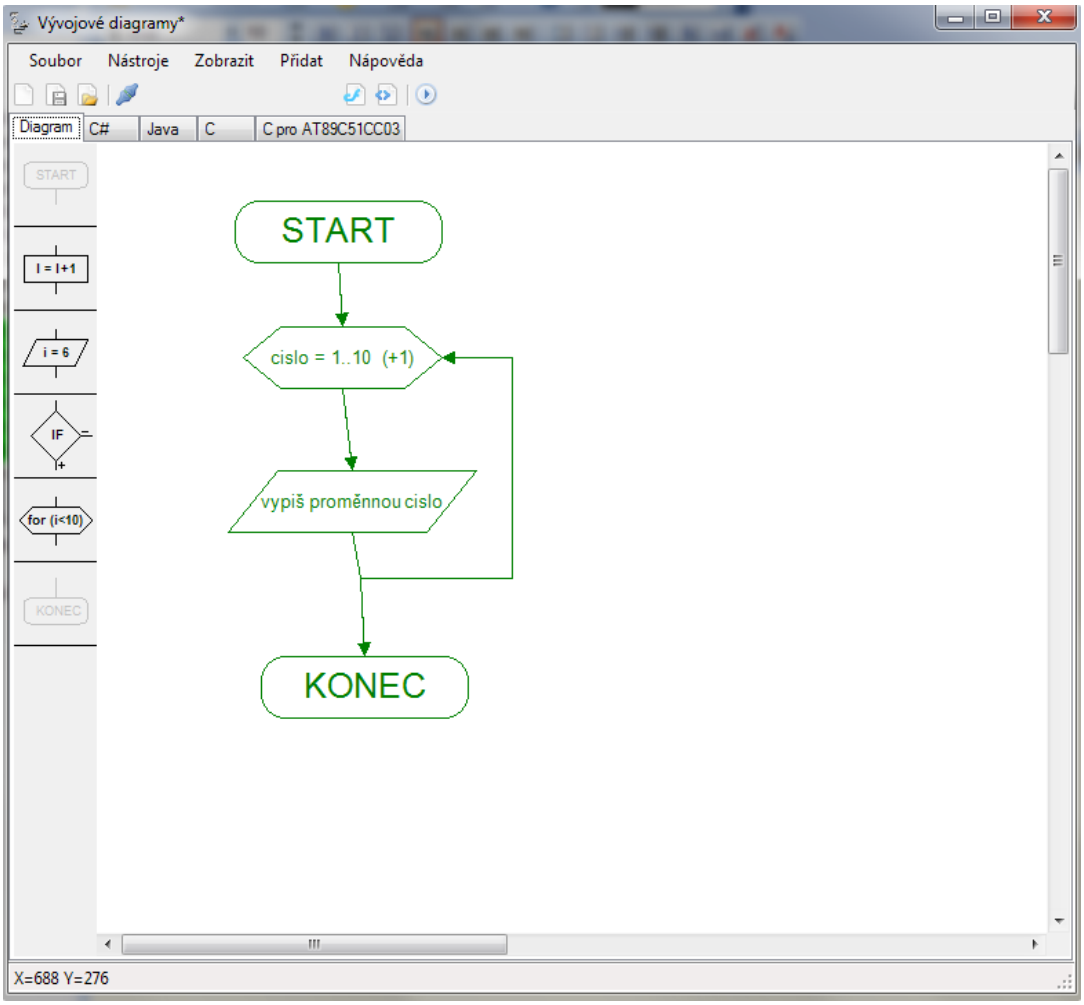
Posledním krokem je spojení symbolů a kontrola vývojového diagramu. Pro spojování symbolů platí tato pravidla:

- Spojnice může začít pouze ve výstupu nějakého symbolu

- Spojnice může končit pouze ve vstupu nějakého symbolu nebo se může spojit s jinou spojnicí
- Výjimkou je symbol pro cyklus, kdy při určení těla cyklu spojnici začínáme v pravém vstupu symbolu pro cyklus
- Spojnice se nemohou větvit, ale mohou se spojovat
- Klikem mimo spojovací body symbolů se vytvoří „koleno“
- Stiskem kolečka myši se smaže poslední koleno, stiskem pravého tlačítka myši se smaže celá spojnice

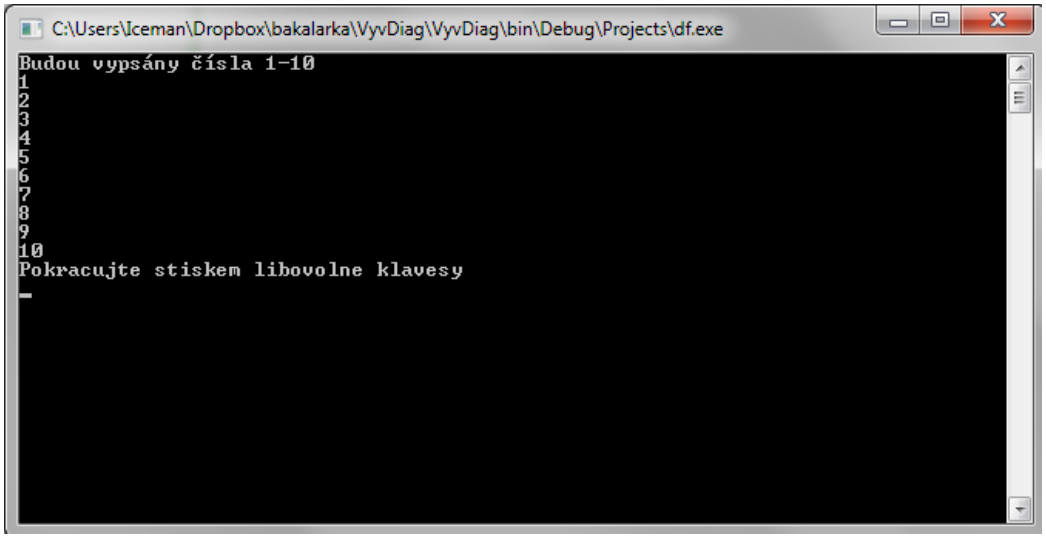
Spojování začneme ve výstupu symbolu *START* (po označení symbolu se zobrazí jako černá tečka uprostřed dolního okraje symbolu). Klikneme na něj a vedeme čáru. Klikem můžeme přidat „*koleno*“. Klikem na vstup symbolu pro cyklus čáru ukončíme. Stejně spojíme symbol pro cyklus a symbol pro výstup a také výstup a konec. Na závěr je nutné definovat tělo cyklu, tj. spojnici kudy se bude cyklus vracet. Začneme kliknutím na pravý vstup symbolu cyklu a spojnici ukončíme spojením se spojnicí mezi symbolem pro výstup a symbolem *KONEC*.

Spojení symbolů můžeme zkontrolovat v menu (Nástroje->Kontrola diagramu) nebo klikem na ikonku kontroly diagramu. Pokud je vše v pořádku, diagram zezelená.



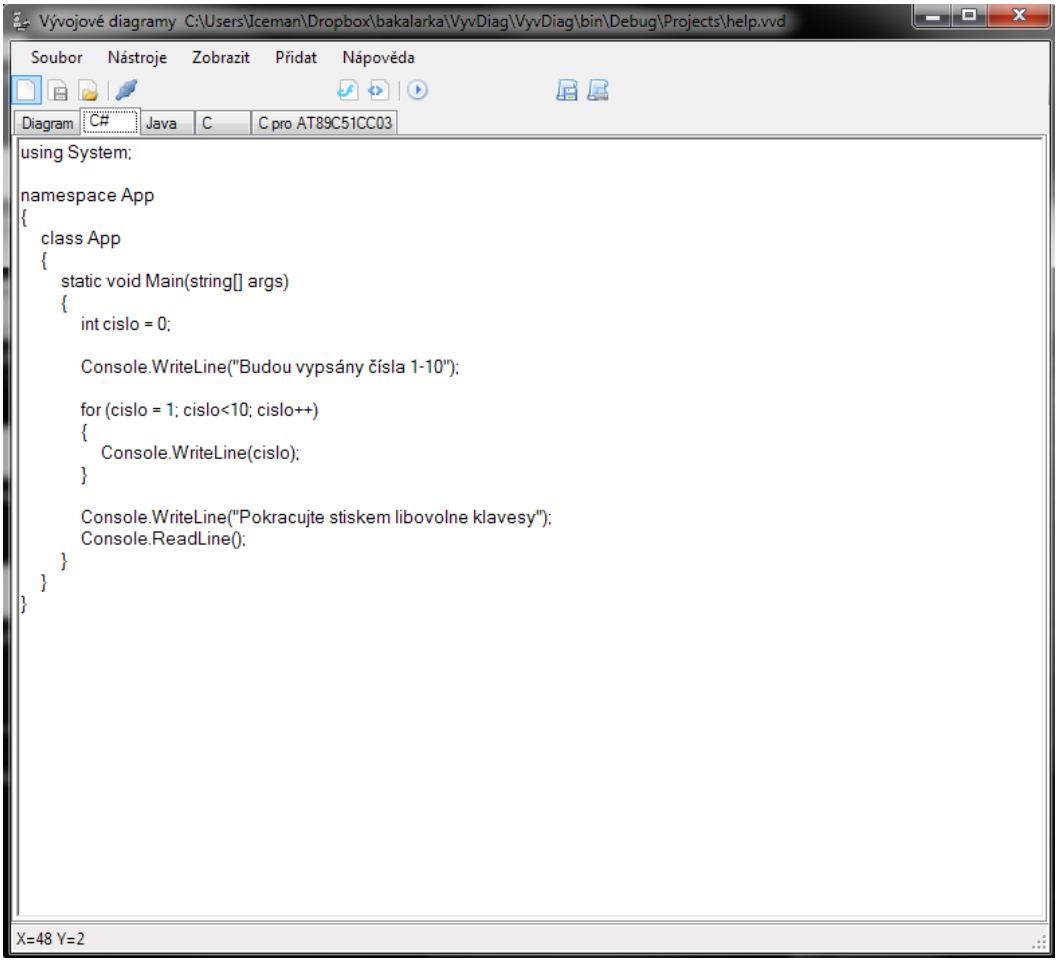
Obrázek 16: Diagram je v pořádku

Pokud jsme se dopracovali až do tohoto bodu, můžeme program spustit. Při spuštění je nutné projekt uložit. Zobrazí se konzole s naším programem.



Obrázek 17: Spuštěný program

Je také možné zobrazit zdrojový kód našeho programu. To provedeme přepnutím na záložku C# v hlavním okně.



Obrázek 18: Zdrojový kód

4.2 Vývojová deska

Vývojový diagram pro desku s mikroprocesorem bude velice podobný jako diagram z předešlého bodu. Jediný podstatný rozdíl je, že nepracujeme s klávesnicí a konzolí, ale s periferiemi na desce. Program se také nekompiluje z C# pomocí *csc.exe*, ale z C pomocí kompilátoru SDCC, jehož cesta musí být vyplněna, stejně jako cesta k packeru *ihx* souborů.

S periferiemi se pracuje pomocí speciálních proměnných, které do programu přidáme pomocí menu (Přidat->Proměnné pro přípravek AT89C51CC03).

Tyto proměnné jsou:

Tlac1 - tlačítko na portu P3_2. 1 - nestisknuto, 0 - stisknuto

Tlac2 - tlačítko na portu P3_3. 1 - nestisknuto, 0 - stisknuto

LED_R - červená dioda na portu P4_2. 1 - zhasnuto, 0 - svítí

LED_Y - žlutá dioda na portu P4_3. 1 - zhasnuto, 0 - svítí

LED_G - zelená dioda na portu P4_4. 1 - zhasnuto, 0 - svítí

REPRO - reproduktor na portu P3_6.

BULB - žárovka na portu P1_2 1 - zhasnuto, 0 - svítí

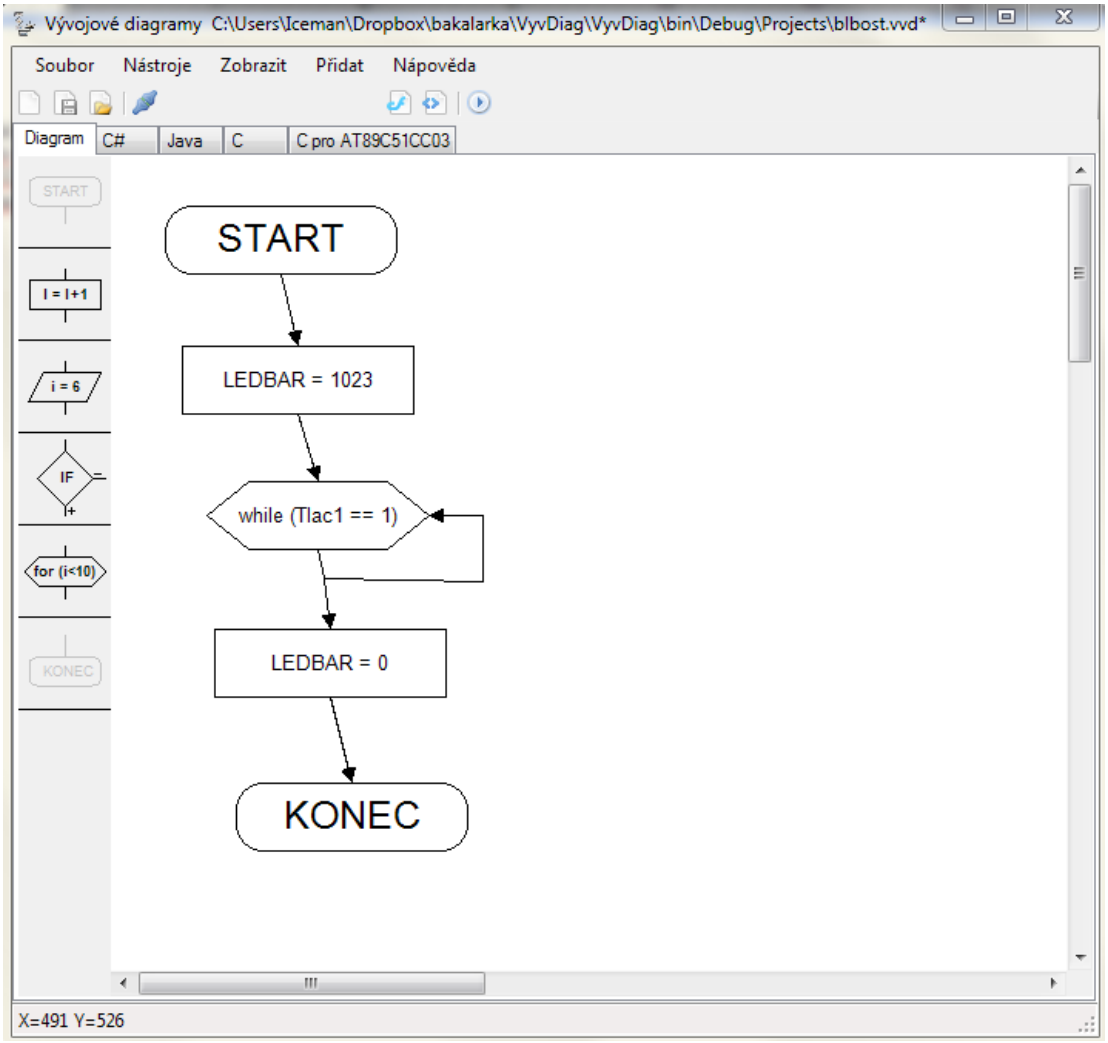
OUTPUT - určuje výstup textu. 0 - displej, 1 - sériová linka

LEDBAR - výstup na *LedBar*. Diody se rozsvítí jako binární reprezentace čísla

STISKNUTO - kontrola, zda je stisknuta nějaká klávesa

CTI_KLAV - znak stisknuté klávesy

Na obrázku je jednoduchý program, který rozsvítí celý *LedBar* a po stisku tlačítka 1 ho zhasne.



Obrázek 19: Program pro vývojovou desku

Rozdíl je také ve způsobu vytvoření spustitelného programu. Program pro desku zkompilujeme tak, že se přepneme do záložky „C pro AT89C51CC03“, v horní liště s ikonami se zpřístupní ikona  a po jejím stisku se ve složce s projektem vygeneruje *hex* soubor, který je možné nahrát do desky.

Příloha B: CD

CD obsahuje zkompilované vývojové prostředí se soubory nutnými pro jeho běh a ukázkovými příklady pro platformu PC i desku Atmel. Dále projekt s aplikací vytvořený v Microsoft Visual Studiu 2008, návod k aplikaci v elektronické podobě a instalační soubory balíku SDCC a .NET Frameworku 3.5.